Memahami dan menerapkan microfrontend pada AWS

AWS Bimbingan Preskriptif

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

AWS Bimbingan Preskriptif: Memahami dan menerapkan microfrontend pada AWS

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Merek dagang dan tampilan dagang Amazon tidak boleh digunakan sehubungan dengan produk atau layanan apa pun yang bukan milik Amazon, dengan cara apa pun yang dapat menyebabkan kebingungan di antara pelanggan, atau dengan cara apa pun yang merendahkan atau mendiskreditkan Amazon. Semua merek dagang lain yang tidak dimiliki oleh Amazon merupakan hak milik masing-masing pemiliknya, yang mungkin atau mungkin tidak terafiliasi, terkait dengan, atau disponsori oleh Amazon.

Table of Contents

Pengantar 1
  Ikhtisar 1
Konsep dasar 6
  Desain berbasis domain 6
  Sistem terdistribusi 8
  Komputasi awan 8
Arsitektur alternatif 10
  Monolit 10
  Aplikasi N-tier 10
  Layanan mikro 11
  Memilih pendekatan untuk kebutuhan Anda 11
Keputusan arsitektur 12
  Batas mikro-frontend 12
    Cara mengiris aplikasi monolitik menjadi frontend mikro 13
  Pendekatan komposisi mikro-frontend 15
    Komposisi di sisi klien 15
    Komposisi sisi tepi 17
    Komposisi di sisi server 17
  Routing dan komunikasi 19
    Perutean 19
    Komunikasi antara mikro-frontend 19
  Kelola dependensi mikro-frontend 20
    Bagikan apa pun, jika memungkinkan 20
    Saat Anda membagikan kode 21
    Status bersama 21
Kerangka kerja dan alat 23
  Pertimbangan kerangka umum 23
Integrasi API - BFF 25
Styling dan CSS 27
  Sistem desain - Pendekatan berbagi sesuatu 27
  CSS yang dienkapsulasi sepenuhnya - Pendekatan berbagi apa-apa 29
  CSS Global Bersama - Pendekatan berbagi semua 29
Organisasi 31
  Pengembangan tangkas 31

Komposisi dan ukuran tim ... 32
DevOps budaya ... 32
Mengatur pengembangan mikro-frontend di beberapa tim ... 34
Men-deploy ... 35
Tata kelola ... 36
Kontrak API ... 36
Interaktivitas silang ... 37
Menyeimbangkan otonomi dan penyelarasan ... 38
Membuat frontend mikro ... 38
End-to-end pengujian untuk mikro-frontend ... 39
Melepaskan mikro-frontend ... 39
Pencatatan log dan pemantauan ... 39
Peringatan ... 40
Bendera fitur ... 41
Penemuan Layanan ... 42
Memisahkan bundel ... 43
Rilis kenari ... 44
Tim platform ... 45
Langkah selanjutnya ... 46
Sumber daya ... 49
Kontributor ... 50
Riwayat dokumen ... 51
Glosarium ... 52
... 52
A ... 53
B ... 56
C ... 58
D ... 61
E ... 65
F ... 67
G ... 69
H ... 70
I ... 71
L ... 74
M ... 75
O ... 79

P 82
Q 85
R 85
D 88
T 92
U 93
V 94
W 94
Z 95
.... xcvii

Memahami dan menerapkan frontend mikro pada AWS

Amazon Web Services (kontributor)

Juli 2024 (sejarah dokumen)

Ketika organisasi berusaha untuk kelincahan dan skalabilitas, arsitektur monolitik konvensional sering menjadi hambatan, menghambat perkembangan dan penyebaran yang cepat. Micro-frontend mengurangi ini dengan memecah antarmuka pengguna yang kompleks menjadi komponen independen yang lebih kecil yang dapat dikembangkan, diuji, dan digunakan secara mandiri. Pendekatan ini meningkatkan efisiensi tim pengembangan dan memfasilitasi kolaborasi antara backend dan frontend, mendorong penyelarasan sistem end-to-end terdistribusi.

Panduan preskriptif ini dirancang untuk membantu para pemimpin TI, pemilik produk, dan arsitek di berbagai domain profesional untuk memahami arsitektur micro-frontend dan membangun aplikasi micro-frontend di Amazon Web Services ().AWS

Ikhtisar

Micro-frontend adalah arsitektur yang dibangun di atas dekomposisi frontend aplikasi menjadi artefak yang dikembangkan dan digunakan secara independen. Ketika Anda membagi frontend besar menjadi artefak perangkat lunak otonom, Anda dapat merangkum logika bisnis dan mengurangi dependensi. Ini mendukung pengiriman kenaikan produk yang lebih cepat dan lebih sering.

Micro-frontend mirip dengan microservices. Bahkan, istilah micro-frontend berasal dari istilah microservice, dan bertujuan untuk menyampaikan gagasan microservice sebagai frontend. Sementara arsitektur microservices biasanya menggabungkan sistem terdistribusi di backend dengan frontend monolitik, micro-frontend adalah layanan frontend terdistribusi mandiri. Layanan ini dapat diatur dengan dua cara:

- Frontend-only, terintegrasi dengan lapisan API bersama di belakangnya yang menjalankan arsitektur layanan mikro
- Full-stack, artinya setiap micro-frontend memiliki implementasi backend sendiri.

Diagram berikut menunjukkan arsitektur layanan mikro tradisional, dengan monolit frontend yang menggunakan gateway API untuk terhubung ke layanan mikro backend.

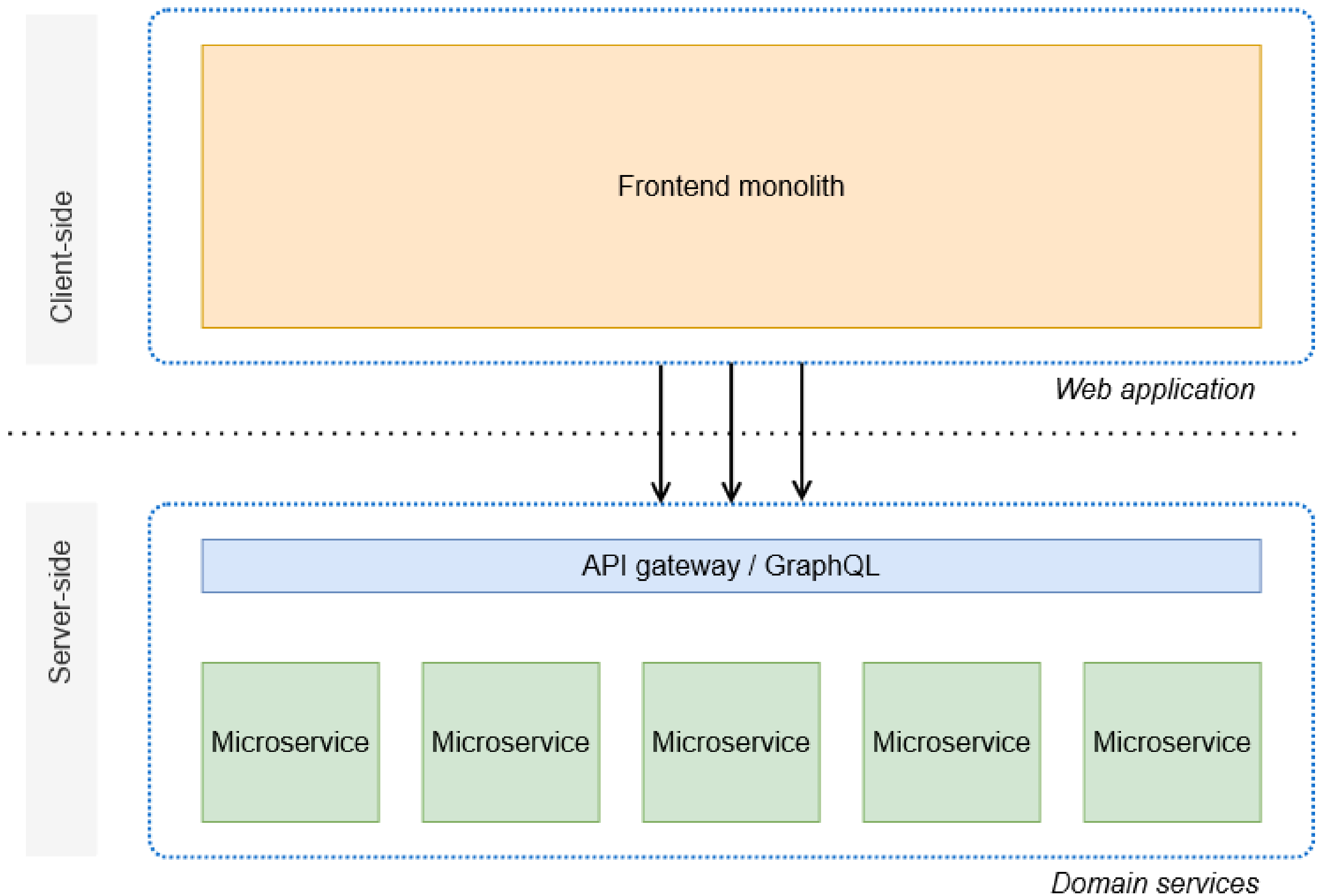
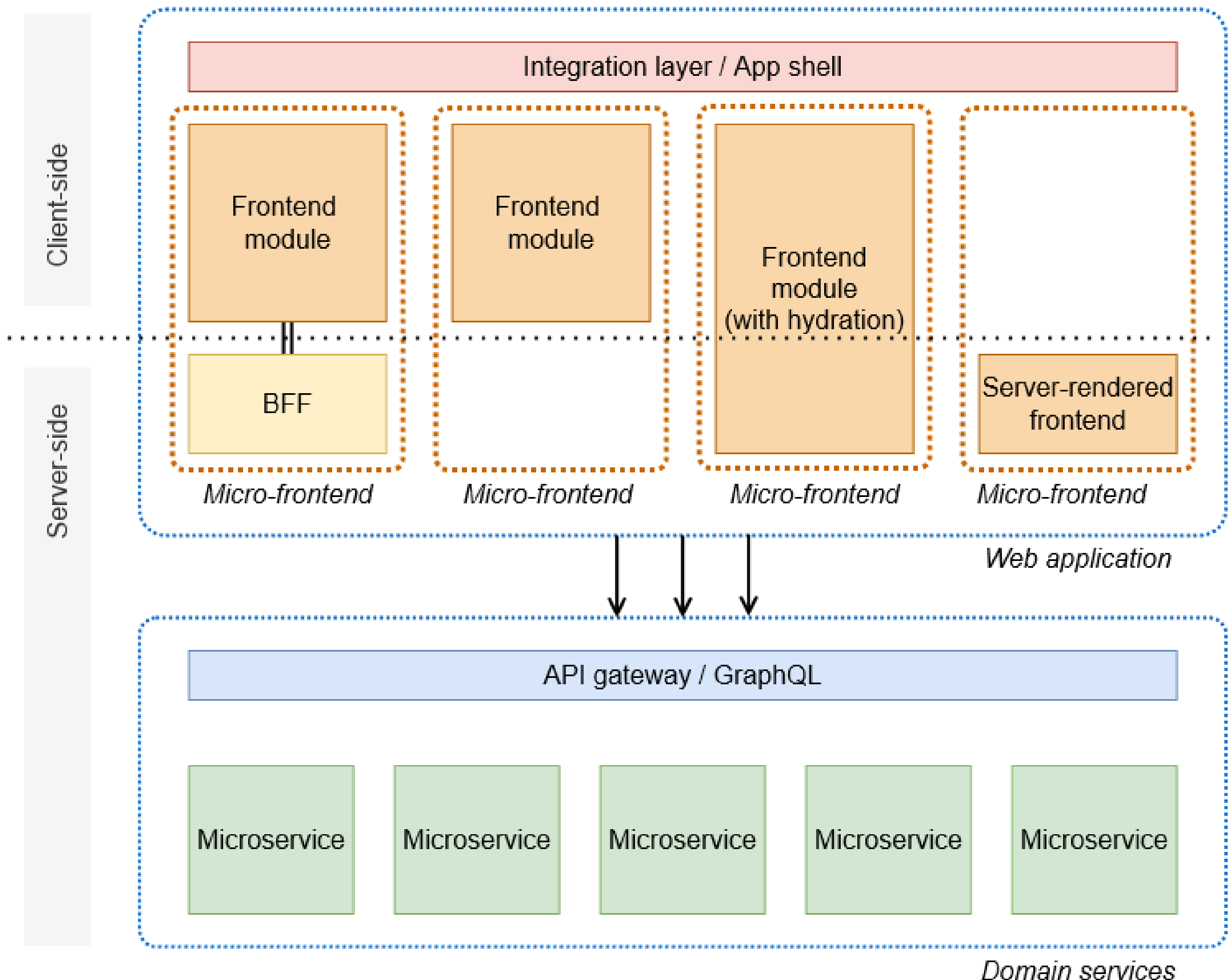


Diagram berikut menunjukkan arsitektur micro-frontend dengan implementasi microservices yang berbeda.



Seperti yang ditunjukkan pada diagram sebelumnya, Anda dapat menggunakan micro-frontend dengan rendering sisi klien atau arsitektur rendering sisi server:

- Frontend mikro yang dirender sisi klien dapat langsung menggunakan APIs ekspos oleh API Gateway terpusat.
- Tim dapat membuat backend-for-frontend (BFF) di dalam konteks terbatas untuk mengurangi obrolan frontend ke arah. APIs
- Di sisi server, mikro-frontend dapat diekspresikan dengan pendekatan sisi server ditambah di sisi klien dengan menggunakan teknik yang disebut hidrasi. Saat halaman dirender oleh browser, halaman terkait akan JavaScript terhidrasi untuk memungkinkan interaksi dengan elemen UI, seperti mengklik tombol.

- Micro-frontend dapat dirender di backend dan menggunakan hyperlink untuk mengarahkan ke bagian baru dari situs web.

Micro-frontend sangat cocok untuk organisasi yang ingin melakukan hal berikut:

- Skala dengan beberapa tim yang mengerjakan proyek yang sama.
- Merangkul desentralisasi pengambilan keputusan, memberdayakan pengembang untuk berinovasi dalam batas-batas sistem yang diidentifikasi.

Pendekatan ini secara signifikan mengurangi beban kognitif pada tim, karena mereka bertanggung jawab atas bagian-bagian tertentu dari sistem. Ini meningkatkan kelincahan bisnis karena modifikasi dapat dilakukan pada satu bagian sistem tanpa mengganggu sisanya.

Micro-frontend adalah pendekatan arsitektur yang berbeda. Meskipun ada berbagai cara untuk membangun frontend mikro, mereka semua memiliki ciri-ciri yang sama:

- Arsitektur micro-frontend terdiri dari beberapa elemen independen. Strukturnya mirip dengan modularisasi yang terjadi dengan layanan mikro di backend.
- Micro-frontend sepenuhnya bertanggung jawab atas implementasi frontend dalam konteks terbatas, yang terdiri dari hal-hal berikut:
  - Antarmuka pengguna
  - Data
  - Negara atau sesi
  - Logika bisnis
  - Aliran

Konteks terbatas adalah sistem yang konsisten secara internal dengan batas-batas yang dirancang dengan hati-hati yang memediasi apa yang bisa masuk dan keluar. Frontend mikro harus berbagi logika bisnis dan data sesedikit mungkin dengan frontend mikro lainnya. Di mana pun berbagi perlu terjadi, itu terjadi melalui antarmuka yang didefinisikan dengan jelas seperti peristiwa khusus atau aliran reaktif. Namun, ketika menyangkut beberapa masalah lintas sektoral seperti sistem desain atau perpustakaan logging, berbagi yang disengaja diterima.

Pola yang direkomendasikan adalah membangun frontend mikro dengan menggunakan tim lintas fungsi. Ini berarti bahwa setiap micro-frontend dikembangkan oleh tim yang sama yang bekerja dari

backend ke frontend. Kepemilikan tim sangat penting, mulai dari pengkodean hingga operasionalisasi sistem dalam produksi.

Panduan ini tidak bermaksud untuk merekomendasikan satu pendekatan tertentu. Sebaliknya, ini membahas berbagai pola, praktik terbaik, trade-off, dan pertimbangan arsitektur dan organisasi.

Konsep dasar

Arsitektur mikro-frontend sangat terinspirasi oleh tiga konsep arsitektur sebelumnya:

- Desain berbasis domain adalah model mental untuk menyusun aplikasi kompleks menjadi domain yang koheren.
- Sistem terdistribusi adalah pendekatan untuk membangun aplikasi sebagai subsistem yang digabungkan secara longgar yang dikembangkan secara independen dan dijalankan pada infrastruktur khusus mereka sendiri.
- Cloud computing adalah pendekatan untuk menjalankan infrastruktur TI sebagai layanan dengan pay-as-you-go model.

Desain berbasis domain

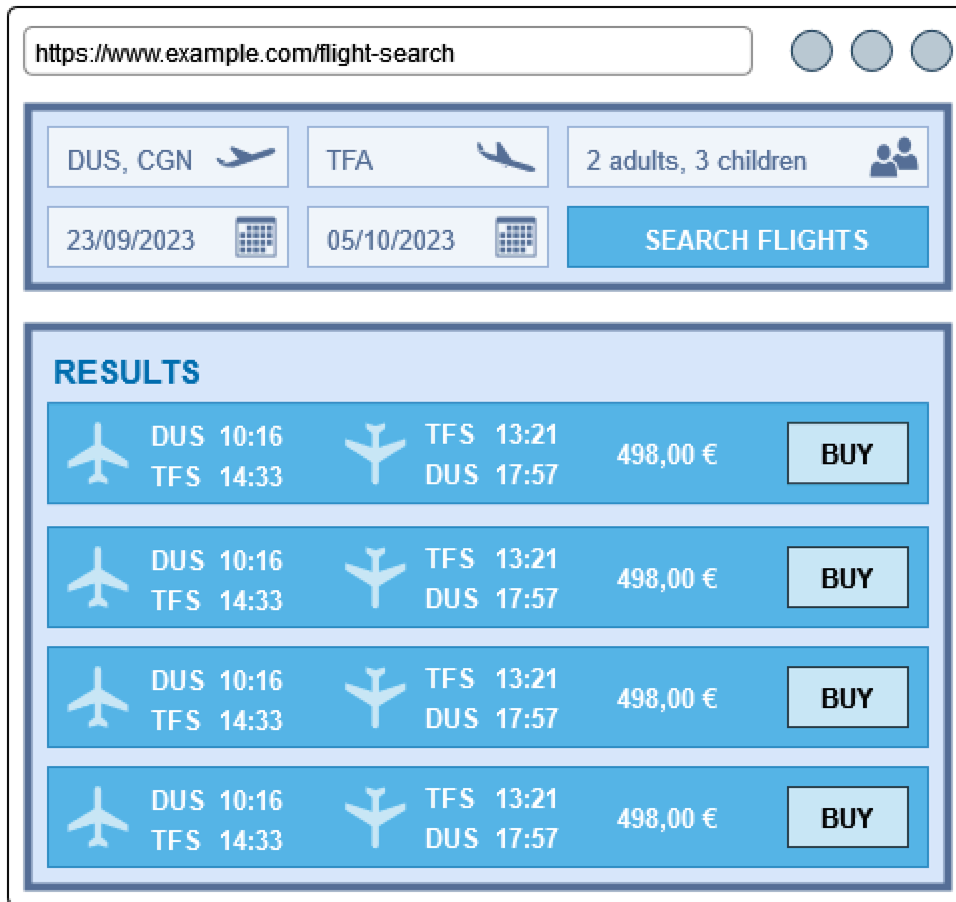
Domain-driven design (DDD) adalah paradigma yang dikembangkan oleh Eric Evans. Dalam bukunya tahun 2003 Domain-Driven Design: Tackling Complexity in the Heart of Software, Evans mendalilkan bahwa pengembangan perangkat lunak harus didorong oleh masalah bisnis daripada masalah teknis. Evans mengusulkan agar proyek TI pertama-tama mengembangkan bahasa di mana-mana yang membantu pakar teknis dan domain untuk menemukan pemahaman bersama. Berdasarkan bahasa itu, mereka dapat merumuskan model realitas bisnis yang saling dipahami.

Jelas seperti pendekatan itu, banyak proyek perangkat lunak mengalami pemutusan hubungan antara bisnis dan TI. Pemutusan hubungan tersebut sering menyebabkan kesalahpahaman yang signifikan, yang menyebabkan pembengkakan anggaran, penurunan kualitas, atau kegagalan proyek.

Evans memperkenalkan beberapa istilah penting lainnya, salah satunya adalah konteks terbatas. Konteks terbatas adalah segmen mandiri dari aplikasi TI besar yang berisi solusi atau implementasi untuk satu masalah bisnis. Aplikasi besar akan terdiri dari beberapa konteks terbatas yang digabungkan secara longgar melalui pola integrasi. Konteks terbatas itu bahkan dapat memiliki dialek mereka sendiri dari bahasa yang ada di mana-mana. Misalnya, pengguna dalam konteks pembayaran aplikasi mungkin memiliki aspek yang berbeda dari pengguna dalam konteks pengiriman karena gagasan pengiriman tidak akan relevan selama pembayaran.

Evans tidak mendefinisikan seberapa kecil atau besar konteks terbatas seharusnya. Ukurannya ditentukan oleh proyek perangkat lunak, dan mungkin berkembang seiring waktu. Indikator yang baik dari batas konteks adalah tingkat kohesi antara entitas (objek domain) dan logika bisnis.

Dalam konteks mikro-frontend, desain berbasis domain dapat diilustrasikan dengan contoh halaman web yang kompleks seperti halaman pemesanan penerbangan.



Pada halaman ini, blok bangunan utama adalah formulir pencarian, panel filter, dan daftar hasil. Untuk mengidentifikasi batas-batas, Anda harus mengidentifikasi konteks fungsional independen. Selain itu, pertimbangkan aspek nonfungsional, seperti penggunaan kembali, kinerja, dan keamanan. Indikator terpenting bahwa "hal-hal yang menjadi milik bersama" adalah pola komunikasi mereka. Jika beberapa elemen dalam arsitektur harus sering berkomunikasi dan bertukar informasi yang kompleks, mereka mungkin berbagi konteks terbatas yang sama.

Elemen UI individual seperti tombol tidak dibatasi konteks, karena mereka tidak independen secara fungsional. Juga, seluruh halaman tidak cocok untuk konteks terbatas, karena dapat dipecah menjadi konteks independen yang lebih kecil. Pendekatan yang masuk akal adalah memperlakukan formulir pencarian sebagai satu konteks terbatas dan memperlakukan daftar hasil sebagai konteks terbatas kedua. Masing-masing dari dua konteks terbatas ini sekarang dapat diimplementasikan sebagai frontend mikro yang terpisah.

Sistem terdistribusi

Untuk memudahkan pemeliharaan dan untuk mendukung kemampuan untuk berkembang, sebagian besar solusi TI nontrivial bersifat modular. Dalam hal ini, modular berarti bahwa sistem TI terdiri dari blok bangunan yang dapat diidentifikasi yang dipisahkan melalui antarmuka untuk mencapai pemisahan masalah.

Selain modular, sistem terdistribusi harus sistem independen dalam hak mereka sendiri. Dalam sistem modular semata, setiap modul idealnya dienkapsulasi dan mengekspos fungsinya melalui antarmuka, tetapi tidak dapat digunakan secara independen atau bahkan berfungsi sendiri. Selain itu, modul umumnya mengikuti siklus hidup yang sama dengan modul lain yang merupakan bagian dari sistem yang sama. Blok bangunan dari sistem terdistribusi, di sisi lain, masing-masing memiliki siklus hidup mereka sendiri. Menerapkan paradigma desain berbasis domain, setiap blok bangunan menangani satu domain bisnis atau subdomain dan hidup dalam konteks terbatasnya sendiri.

Ketika sistem terdistribusi berinteraksi selama waktu pembuatan, pendekatan umum adalah mengembangkan mekanisme untuk mengidentifikasi masalah dengan cepat. Misalnya, Anda mungkin mengadopsi bahasa yang diketik dan berinvestasi besar-besaran dalam pengujian unit. Beberapa tim dapat berkolaborasi dalam pengembangan dan pemeliharaan modul, sering didistribusikan sebagai pustaka untuk digunakan sistem dengan alat seperti npm, Apache Maven, dan pip. NuGet

Selama runtime, sistem terdistribusi yang berinteraksi biasanya dimiliki oleh tim individu. Mengkonsumsi dependensi menyebabkan kompleksitas operasional karena penanganan kesalahan, keseimbangan kinerja, dan keamanan. Investasi dalam pengujian integrasi dan observabilitas sangat penting untuk mengurangi risiko.

Contoh paling populer dari sistem terdistribusi saat ini adalah layanan mikro. Dalam arsitektur microservice, layanan backend digerakkan oleh domain (bukan didorong oleh masalah teknis seperti UI atau otentikasi) dan dimiliki oleh tim otonom. Micro-frontend berbagi prinsip yang sama, memperluas cakupan solusi ke frontend.

Komputasi awan

Cloud computing adalah cara membeli infrastruktur TI sebagai layanan dengan pay-as-you-go model alih-alih membangun pusat data Anda sendiri dan membeli perangkat keras untuk mengoperasikannya di tempat. Cloud computing menawarkan beberapa keuntungan:

- Organisasi Anda memperoleh kelincahan bisnis yang signifikan dengan dapat bereksperimen dengan teknologi baru tanpa harus membuat komitmen keuangan jangka panjang yang besar di muka.
- Dengan menggunakan penyedia cloud seperti AWS, organisasi Anda dapat mengakses portofolio luas layanan pemeliharaan rendah dan sangat terintegrasi (seperti gateway API, database, orkestrasi kontainer, dan kemampuan cloud). Akses ke layanan ini membebaskan staf Anda untuk fokus pada pekerjaan yang membedakan organisasi Anda dari pesaing.
- Ketika organisasi Anda siap untuk meluncurkan solusi secara global, Anda dapat menerapkan solusi untuk infrastruktur cloud di seluruh dunia.

Cloud computing mendukung micro-frontend dengan menyediakan infrastruktur yang sangat terkelola. Ini membuat end-to-end kepemilikan lebih mudah bagi tim lintas fungsi. Sementara tim harus memiliki pengetahuan operasi yang kuat, tugas manual penyediaan infrastruktur, pembaruan sistem operasi, dan jaringan akan menjadi gangguan.

Karena frontend mikro hidup dalam konteks terbatas, tim dapat memilih layanan yang paling cocok untuk menjalankannya. Misalnya, tim dapat memilih antara fungsi cloud dan wadah untuk komputasi, dan mereka dapat memilih antara berbagai ragam database SQL dan NoSQL atau cache dalam memori. Tim bahkan dapat membangun frontend mikro mereka pada toolkit yang sangat terintegrasi seperti AWS Amplify, yang dilengkapi dengan blok bangunan yang telah dikonfigurasi sebelumnya untuk infrastruktur tanpa server.

Membandingkan mikro-frontend dengan arsitektur alternatif

Seperti semua strategi arsitektur, keputusan untuk mengadopsi frontend mikro harus didasarkan pada kriteria evaluasi yang dipandu oleh prinsip-prinsip organisasi Anda. Micro-frontend memiliki kelebihan dan kekurangan. Jika organisasi Anda memutuskan untuk menggunakan frontend mikro, Anda harus memiliki strategi untuk mengatasi tantangan sistem terdistribusi

Saat memilih arsitektur aplikasi, alternatif paling populer untuk frontend mikro adalah monolit, aplikasi n-tier, dan layanan mikro dalam kombinasi dengan frontend aplikasi satu halaman (SPA). Ini semua adalah pendekatan yang valid, dan masing-masing memiliki kelebihan dan kekurangan.

Monolit

Aplikasi kecil yang tidak perlu sering diubah dapat dikirimkan dengan sangat cepat sebagai monolit. Bahkan dalam situasi di mana pertumbuhan yang signifikan diharapkan, monolit adalah langkah pertama yang alami. Kemudian, monolit dapat dipensiunkan atau difaktorkan ulang menjadi struktur yang lebih fleksibel. Dengan memulai dengan monolit, organisasi Anda dapat pergi ke pasar, mendapatkan umpan balik pelanggan, dan meningkatkan produk lebih cepat.

Namun, aplikasi monolitik cenderung terdegradasi jika tidak dipelihara dengan hati-hati atau karena basis kode tumbuh dalam ukuran dari waktu ke waktu. Ketika beberapa tim secara signifikan berkontribusi pada basis kode yang sama, mereka jarang berkontribusi pada pemeliharaan dan operasinya. Hal ini mengakibatkan ketidakseimbangan tanggung jawab, yang berdampak pada kecepatan dan menyebabkan inefisiensi. Pada saat yang sama, kopling yang tidak disengaja antara modul monolit menyebabkan efek samping yang tidak diinginkan saat basis kode berkembang. Efek samping tersebut dapat mengakibatkan malfungsi dan pemadaman.

Aplikasi N-tier

Aplikasi yang lebih kompleks yang memiliki kecepatan evolusi yang relatif statis dapat dibangun sebagai arsitektur tiga tingkat (presentasi, aplikasi, data), dengan lapisan REST atau GraphQL antara frontend dan backend. Ini jauh lebih fleksibel, dan tim untuk tingkatan yang berbeda dapat berkembang secara independen sampai batas tertentu. Kerugian dari aplikasi n-tier adalah jauh lebih sulit untuk menerapkan fungsionalitas. Frontend dan backend dipisahkan melalui kontrak API, jadi perubahan yang melanggar harus diterapkan bersama atau API harus dibuat versi.

Pertimbangkan skenario umum berikut: Jika merilis fitur baru memerlukan perubahan skema data, mungkin perlu berhari-hari bagi pemilik produk untuk menyetujui serangkaian fungsi dengan tim

frontend. Kemudian tim frontend akan meminta tim backend untuk mengembangkan dan melepaskan fungsionalitas di pihak mereka. Tim backend akan bekerja dengan pemilik data untuk merilis pembaruan skema database. Selanjutnya, tim backend akan merilis versi baru API, sehingga tim frontend dapat mengembangkan dan merilis perubahan mereka. Dalam skenario ini, menyebarkan semua perubahan pada produksi mungkin memakan waktu berminggu-minggu atau bahkan berbulan-bulan, karena setiap tim memiliki backlog, prioritas, dan mekanisme sendiri seputar pengembangan, pengujian, dan pelepasan perubahan.

Layanan mikro

Dalam arsitektur microservices, backend didekomposisi menjadi layanan kecil, masing-masing menangani masalah bisnis tertentu dalam konteks terbatas. Setiap layanan mikro juga sangat dipisahkan dari layanan lain dengan mengekspos kontrak antarmuka yang jelas.

Perlu disebutkan bahwa konteks terbatas dan kontrak antarmuka juga harus ada dalam monolit yang dibuat dengan baik dan arsitektur n-tier. Namun, dalam arsitektur microservices, komunikasi terjadi melalui jaringan, biasanya protokol HTTP, dan layanan memiliki infrastruktur runtime khusus. Ini mendukung pengembangan independen, pengiriman, dan pengoperasian setiap layanan backend.

Memilih pendekatan untuk kebutuhan Anda

Monolit dan arsitektur n-tier menggabungkan beberapa masalah domain menjadi satu artefak teknis. Hal ini membuat aspek-aspek seperti dependensi dan aliran data internal mudah dikelola, tetapi membuat pengiriman fungsionalitas baru menjadi lebih sulit. Untuk mempertahankan basis kode yang koheren, tim sering menginvestasikan waktu dalam refactoring dan decoupling karena basis kode besar yang harus mereka tangani.

Aplikasi yang dikembangkan oleh beberapa tim mungkin tidak memerlukan kompleksitas tambahan yang datang dengan pindah ke frontend mikro. Ini terutama benar jika tim tidak membayar penalti kopling tinggi dan waktu tunggu yang lama untuk melepaskan perubahan.

Singkatnya, arsitektur yang lebih kompleks dan terdistribusi seringkali merupakan pilihan yang tepat untuk aplikasi yang kompleks dan bergerak cepat. Untuk aplikasi kecil hingga menengah, arsitektur terdistribusi tidak selalu lebih unggul dari yang monolitik, terutama jika aplikasi tidak akan berkembang secara dramatis dalam waktu singkat.

Keputusan arsitektur dalam frontend mikro

Tim yang menerapkan pola arsitektur micro-frontend untuk aplikasi mereka harus membuat beberapa keputusan tentang arsitektur sejak dini:

- Identifikasi frontend mikro dan definisi batas
- Menyusun halaman dan tampilan dengan frontend mikro
- Perutean, manajemen negara, dan komunikasi di seluruh frontend mikro
- Mengelola dependensi untuk masalah lintas sektoral

Bagian berikut mencakup topik-topik ini secara lebih mendalam.

Saat membuat keputusan arsitektur, penting untuk memiliki metrik yang benar dan memahami karakteristik aplikasi pola penggunaan, dan trade-off. Misalnya, situs e-commerce memiliki karakteristik dan pola penggunaan yang berbeda dibandingkan dengan alat pengeditan video atau dasbor observabilitas.

Aplikasi yang menghadap publik dengan lalu lintas tinggi dan kedalaman sesi pendek dapat dioptimalkan untuk metrik pemuatan halaman awal seperti Time to Interactive (TTI) dan First Contentful Paint (FCP). Sebaliknya, aplikasi tempat pengguna masuk di awal hari mereka dan terus berinteraksi sepanjang hari mungkin dioptimalkan untuk pengalaman dalam aplikasi. Tim aplikasi mungkin mengoptimalkan metrik First Input Delay (FID) setelah setiap navigasi, bukan pemuatan halaman awal.

Situs web publik harus melayani berbagai lingkungan browser. Aplikasi perusahaan dengan kendala yang diketahui pada lingkungan klien dapat mengoptimalkan komposisi mikro-frontend mereka sesuai dengan kendala mereka.

Tidak ada pilihan tunggal yang tepat untuk keputusan arsitektur. Memahami trade-off, konteks di mana bisnis beroperasi, pola penggunaan, dan metrik untuk memandu keputusan yang sesuai untuk setiap aplikasi individu.

Mengidentifikasi batas mikro-frontend

Untuk meningkatkan otonomi tim, kemampuan bisnis yang disediakan oleh aplikasi dapat didekomposisi menjadi beberapa frontend mikro dengan ketergantungan minimal satu sama lain.

Mengikuti metodologi DDD yang dibahas sebelumnya, tim dapat memecah domain aplikasi menjadi subdomain bisnis dan konteks terbatas. Tim otonom kemudian dapat memiliki fungsionalitas konteks terbatas mereka dan memberikan konteks tersebut sebagai frontend mikro.

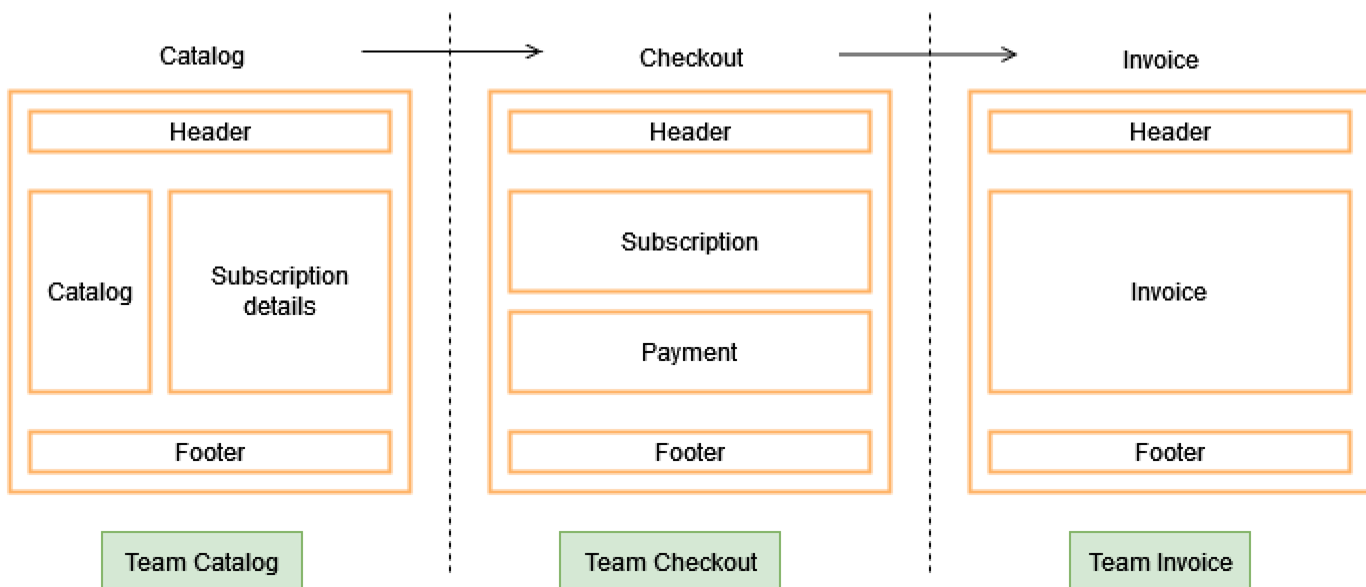
Konteks terbatas yang terdefinisi dengan baik harus meminimalkan tumpang tindih fungsional dan kebutuhan akan komunikasi runtime lintas konteks. Komunikasi yang diperlukan dapat diimplementasikan dengan metode yang digerakkan oleh peristiwa. Ini tidak berbeda dengan arsitektur berbasis peristiwa untuk pengembangan layanan mikro.

Aplikasi yang dirancang dengan baik juga harus mendukung pengiriman ekstensi future oleh tim baru untuk memberikan pengalaman yang konsisten bagi pelanggan.

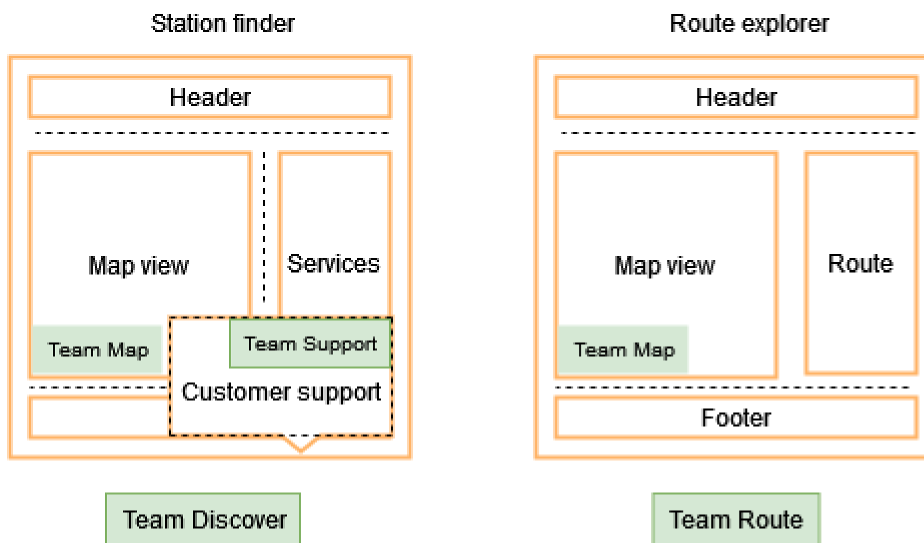
Cara mengiris aplikasi monolitik menjadi frontend mikro

Bagian Ikhtisar menyertakan contoh mengidentifikasi konteks fungsional independen pada halaman web. Beberapa pola untuk memisahkan fungsionalitas pada antarmuka pengguna muncul.

Misalnya, ketika domain bisnis membentuk tahapan perjalanan pengguna, pemisahan vertikal pada frontend dapat diterapkan, di mana kumpulan tampilan dalam perjalanan pengguna disampaikan sebagai frontend mikro. Diagram berikut menunjukkan pemisahan vertikal, di mana langkah-langkah Katalog, Checkout, dan Faktur disampaikan oleh tim terpisah sebagai frontend mikro yang terpisah.



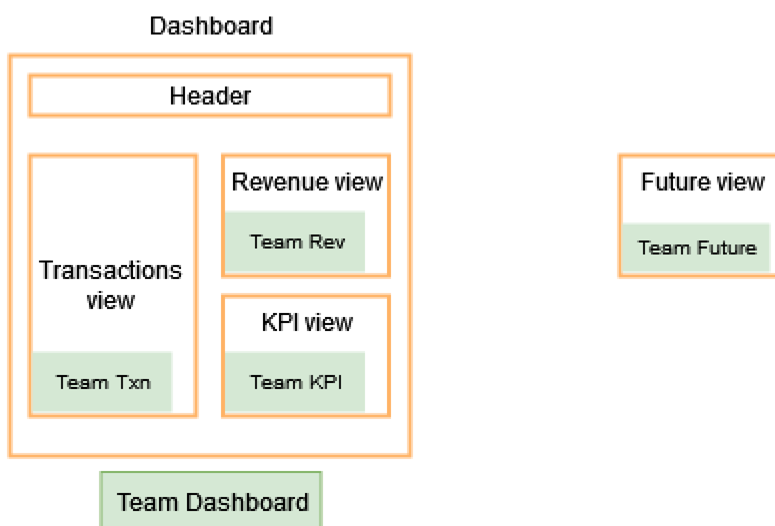
Untuk beberapa aplikasi, pemisahan vertikal saja mungkin tidak cukup. Misalnya, beberapa fungsi mungkin perlu disediakan dalam banyak tampilan. Untuk aplikasi ini, Anda dapat menerapkan split campuran. Diagram berikut menunjukkan solusi split campuran di mana frontend mikro untuk pencari Stasiun dan penjelajah Rute keduanya menggunakan fungsionalitas tampilan Peta.



Aplikasi tipe portal atau tipe dasbor biasanya menyatukan kemampuan frontend dalam satu tampilan. Dalam jenis aplikasi ini, setiap widget dapat dikirimkan sebagai micro-frontend, dan aplikasi hosting mendefinisikan kendala dan antarmuka yang harus diimplementasikan oleh micro-frontend.

Pendekatan ini menyediakan mekanisme untuk frontend mikro untuk menangani masalah seperti ukuran viewport, penyedia otentikasi, pengaturan konfigurasi, dan metadata. Jenis aplikasi ini mengoptimalkan untuk ekstensibilitas. Fitur baru dapat dikembangkan oleh tim baru untuk meningkatkan kemampuan dasbor.

Diagram berikut menunjukkan aplikasi dasbor yang dikembangkan oleh tiga tim individu yang merupakan bagian dari Team Dashboard.



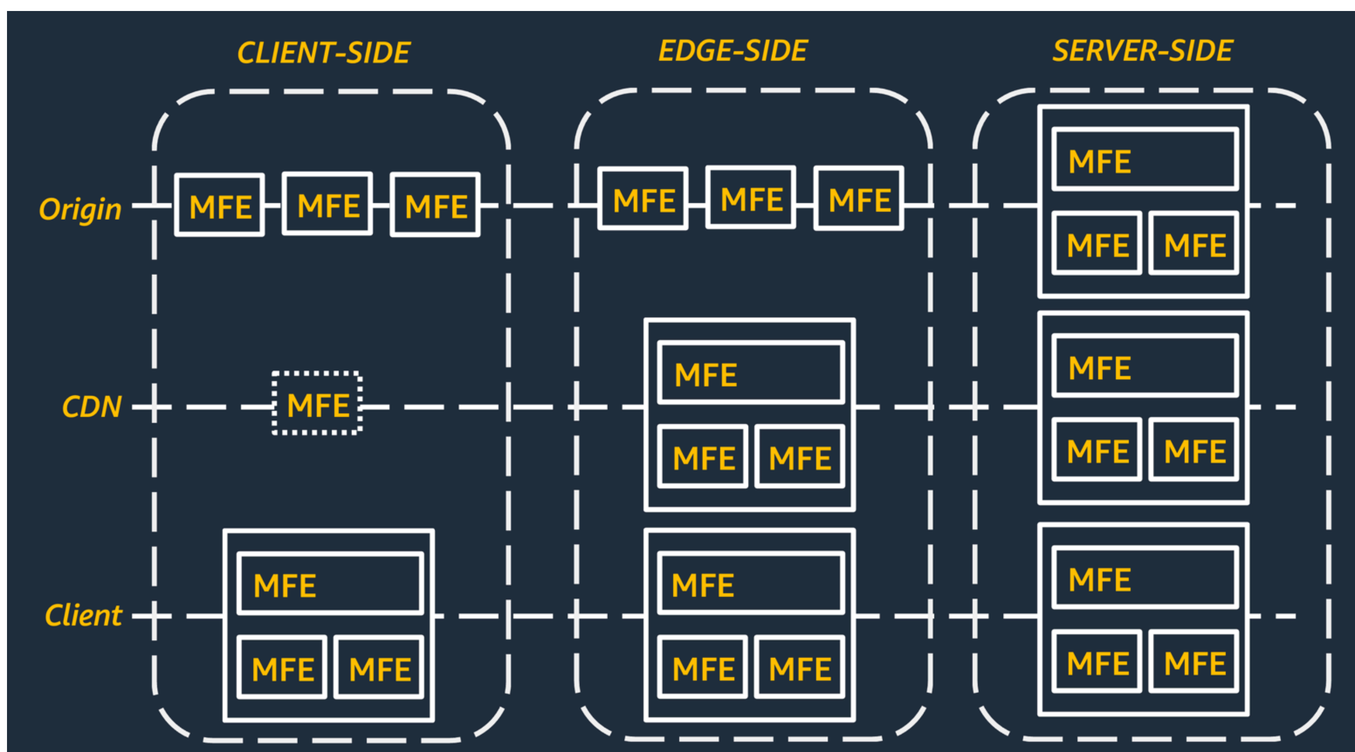
Dalam diagram, tampilan future mewakili fitur baru yang dikembangkan oleh tim baru untuk menskalakan Dasbor Tim dan kemampuan dasbor.

Aplikasi portal dan dasbor biasanya menyusun fungsionalitas dengan menggunakan split campuran di UI. Micro-frontend dapat dikonfigurasi dengan pengaturan yang terdefinisi dengan baik, termasuk batasan posisi dan ukuran.

Menyusun halaman dan tampilan dengan frontend mikro

Anda dapat menyusun tampilan aplikasi dengan komposisi sisi klien, komposisi sisi tepi, dan komposisi sisi server. Pola komposisi memiliki karakteristik yang berbeda dalam hal keterampilan tim yang diperlukan, toleransi kesalahan, kinerja, dan perilaku cache.

Diagram berikut menunjukkan bagaimana komposisi terjadi pada lapisan sisi klien, sisi tepi, dan sisi server dari arsitektur mikro-frontend.



Lapisan sisi klien, sisi tepi dan sisi server dibahas di bagian berikut.

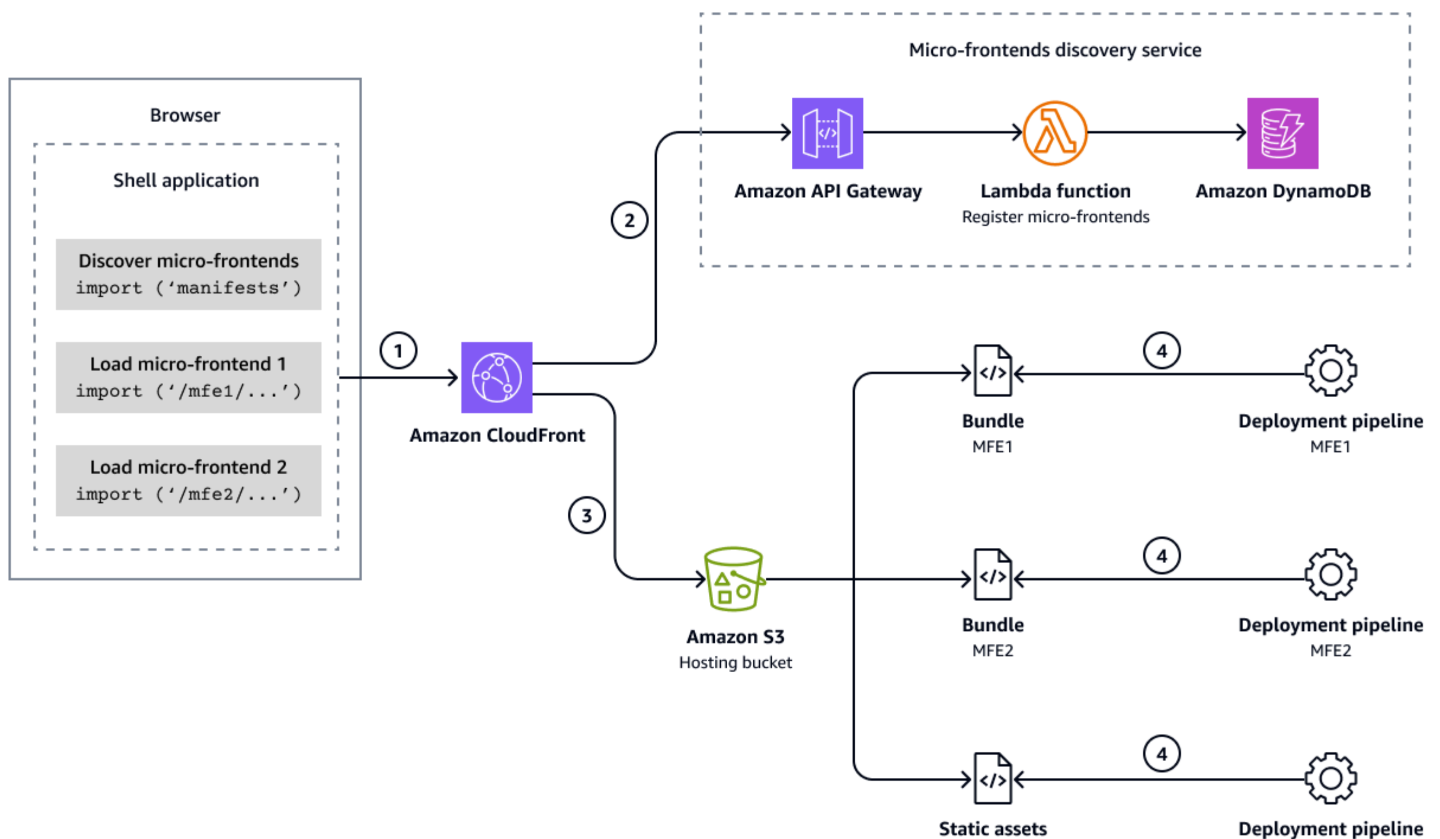
Komposisi di sisi klien

Secara dinamis memuat dan menambahkan micro-frontend sebagai fragmen Document Object Model (DOM) pada klien (browser atau tampilan web seluler). Artefak mikro-frontend, seperti

JavaScript atau file CSS, dapat dimuat dari jaringan pengiriman konten (CDNs) untuk mengurangi latensi. Komposisi sisi klien membutuhkan yang berikut:

- Tim untuk memiliki dan memelihara aplikasi shell atau kerangka kerja micro-frontend untuk mengaktifkan penemuan, pemuatan, dan rendering komponen micro-frontend saat runtime di browser
- Tingkat keterampilan tinggi dalam teknologi frontend seperti HTML, CSS, dan JavaScript, dan pemahaman mendalam tentang lingkungan browser
- Optimalisasi jumlah yang JavaScript dimuat dalam halaman, dan disiplin untuk menghindari bentrokan namespace global

Diagram berikut menunjukkan contoh AWS arsitektur untuk komposisi sisi klien tanpa server.



Komposisi sisi klien terjadi di lingkungan browser melalui aplikasi shell. Diagram menunjukkan rincian berikut:

1. Setelah aplikasi shell dimuat, aplikasi ini membuat permintaan awal ke Amazon CloudFront untuk menemukan frontend mikro yang akan dimuat melalui titik akhir manifes.

2. Manifes berisi informasi tentang setiap mikro-frontend (misalnya, nama, URL, versi, dan perilaku fallback). Manifestasi dilayani oleh layanan penemuan mikro-frontend. Dalam diagram, layanan penemuan ini diwakili oleh Amazon API Gateway, sebuah AWS Lambda fungsi, dan Amazon DynamoDB. Aplikasi shell menggunakan informasi manifes untuk meminta frontend mikro individu untuk menyusun halaman dalam tata letak tertentu.
3. Setiap bundel micro-frontend terdiri dari file statis (seperti JavaScript, CSS, dan HTML). File-file tersebut di-host di bucket Amazon Simple Storage Service (Amazon S3) dan dilayani. CloudFront
4. Tim dapat menerapkan versi baru dari frontend mikro mereka dan memperbarui informasi manifes dengan menggunakan pipeline penerapan yang mereka miliki.

Komposisi sisi tepi

Gunakan teknik transklusi seperti Edge Side Included (ESI) atau Server Side Included (SSI) yang didukung oleh beberapa CDNs dan proxy di depan server asal untuk membuat halaman sebelum mengirimkannya melalui kabel ke klien. ESI membutuhkan yang berikut:

- CDN dengan kemampuan ESI, atau penyebaran proxy di depan frontend mikro sisi server. Implementasi proxy seperti HAProxy, Varnish, dan NGINX mendukung SSI.
- Pemahaman tentang penggunaan dan keterbatasan implementasi ESI dan SSI.

Tim yang memulai aplikasi baru biasanya tidak memilih komposisi sisi tepi untuk pola komposisi mereka. Namun, pola ini mungkin menyediakan jalur untuk aplikasi lama yang mengandalkan transklusi.

Komposisi di sisi server

Gunakan server asal untuk membuat halaman sebelum di-cache di tepi. Hal ini dapat dilakukan dengan teknologi tradisional, seperti PHP, Jakarta Server Pages (JSP), atau template library, untuk menyusun halaman dengan memasukkan fragmen dari micro-frontend. Anda juga dapat menggunakan JavaScript kerangka kerja, seperti Next.js, yang berjalan di server untuk membuat halaman di server dengan rendering sisi server (SSR).

Setelah halaman dirender di server, mereka dapat di-cache CDNs untuk mengurangi latensi. Ketika versi baru micro-frontend dikerahkan, halaman harus dirender ulang, dan cache harus diperbarui untuk mengirimkan versi terbaru kepada pelanggan.

Komposisi sisi server memerlukan pemahaman mendalam tentang lingkungan server untuk menetapkan pola penerapan, penemuan mikro-frontend sisi server, dan manajemen cache.

Diagram berikut menunjukkan komposisi sisi server.

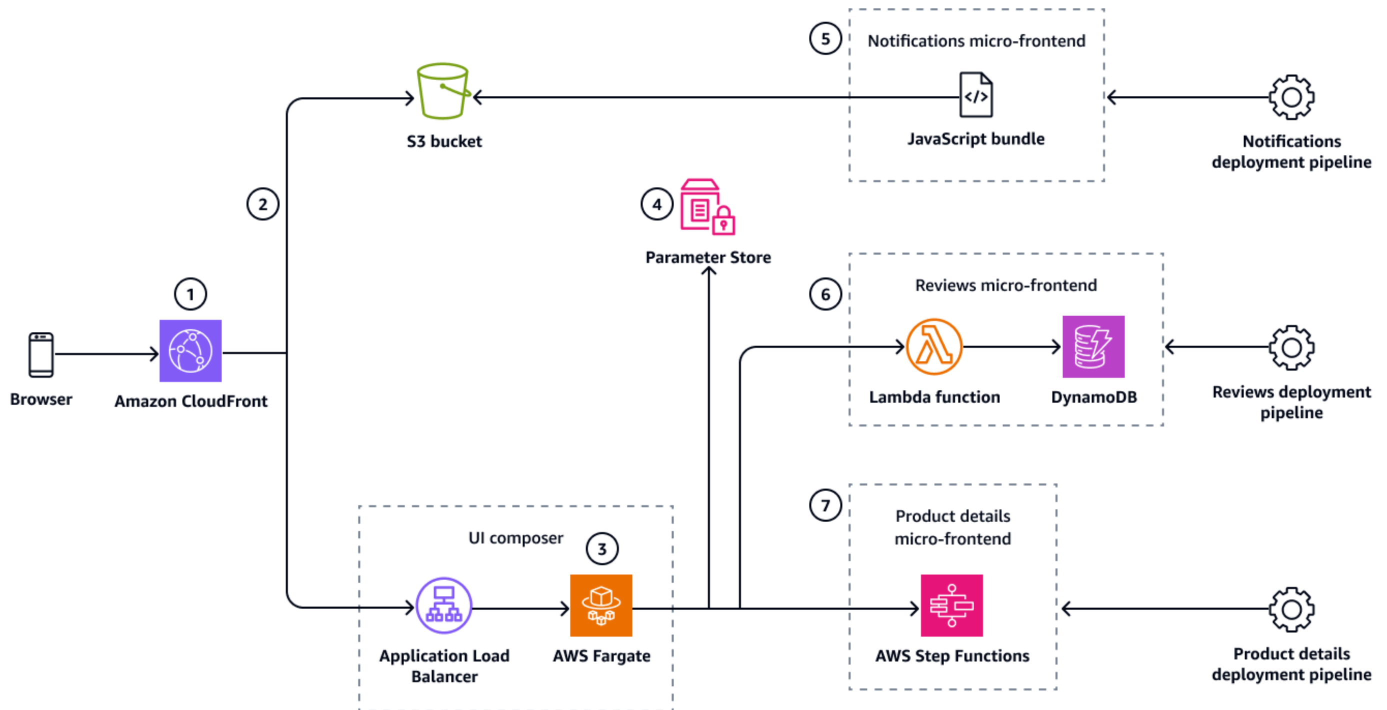


Diagram mencakup komponen dan proses berikut:

1. Amazon CloudFront menyediakan titik masuk unik ke aplikasi. Distribusi memiliki dua asal: yang pertama untuk file statis dan yang kedua untuk komposer UI.
2. File statis di-host di bucket Amazon S3. Mereka dikonsumsi oleh browser dan komposer UI untuk template HTML.
3. Komposer UI berjalan pada cluster kontainer di AWS Fargate. Dengan solusi kontainerisasi, Anda dapat menggunakan kemampuan streaming dan rendering multithreaded jika diperlukan.
4. Parameter Store, kemampuan AWS Systems Manager, digunakan sebagai sistem penemuan mikro-frontend dasar. Kemampuan ini menyediakan penyimpanan nilai kunci yang digunakan oleh komposer UI untuk mengambil endpoint mikro-frontend untuk dikonsumsi.
5. Notifikasi micro-frontend menyimpan JavaScript bundel yang dioptimalkan di bucket S3. Ini dirender pada klien karena harus bereaksi terhadap interaksi pengguna.
6. Ulasan micro-frontend disusun oleh fungsi Lambda, dan ulasan pengguna disimpan di DynamoDB. Ulasan micro-frontend diberikan sepenuhnya di sisi server, dan menghasilkan fragmen HTML.

7. Detail produk micro-frontend adalah micro-frontend kode rendah yang digunakan. AWS Step Functions Alur Kerja Ekspres dapat dipanggil secara sinkron, dan berisi logika untuk merender fragmen HTML dan lapisan caching.

Untuk informasi lebih lanjut tentang komposisi sisi server, lihat posting blog Server-side rendering micro-frontend - arsitekturnya.

Perutean dan komunikasi di seluruh frontend mikro

Opsi perutean tergantung pada pendekatan komposisi. Komunikasi dapat dioptimalkan dengan mengurangi kopling antara komponen frontend.

Perutean

Aplikasi yang menggunakan komposisi sisi klien dengan split vertikal dapat menggunakan perutean sisi server (aplikasi multihalaman) atau routing sisi klien (aplikasi satu halaman). Jika mereka menggunakan split campuran untuk komposisi UI, perutean sisi klien diperlukan untuk mendukung hierarki perutean yang lebih dalam dari mikro-frontend pada halaman.

Aplikasi yang menggunakan komposisi sisi tepi dan komposisi sisi server lebih selaras dengan perutean sisi server, atau perutean dengan komputasi tepi seperti Lambda @Edge dengan Amazon. CloudFront

Komunikasi antara mikro-frontend

Dengan arsitektur micro-frontend, kami merekomendasikan untuk mengurangi kopling antara komponen frontend. Salah satu pendekatan untuk mengurangi kopling adalah dengan menjauh dari panggilan fungsi sinkron ke pesan asinkron.

Runtime browser dan interaksi pengguna bersifat asinkron. Acara dapat dipertukarkan antara produsen dan konsumen melalui pesan. Peristiwa menyediakan antarmuka yang terdefinisi dengan baik untuk komunikasi di seluruh frontend mikro.

Jika Anda mengikuti praktik DDD untuk mengidentifikasi konteks terbatas Anda untuk frontend mikro, langkah selanjutnya adalah mengidentifikasi peristiwa yang harus dikomunikasikan melintasi batas.

Mekanisme pesan untuk acara dapat berupa peristiwa DOM asli (`CustomEvents`), pemancar JavaScript peristiwa, atau pustaka aliran reaktif yang disediakan oleh tim platform. Micro-frontend mempublikasikan acara dan berlangganan acara yang relevan untuk konteks terbatas mereka.

Dengan metode ini, penerbit dan pelanggan tidak perlu saling menyadari. Kontrak adalah definisi acara. Untuk representasi visual ini, lihat bagian Berkomunikasi dengan peristiwa dari diagram konteks Terbatas dengan arsitektur peristiwa.

Mengelola dependensi untuk masalah lintas sektoral

Manajemen ketergantungan sadar sangat penting untuk keberhasilan arsitektur terdistribusi seperti mikro-frontend. Manajemen ketergantungan adalah salah satu bagian yang paling menantang dari pengembangan mikro-frontend.

Dalam arsitektur mikro-frontend, dua aspek penting dari manajemen ketergantungan adalah penalti kinerja dari mentransfer artefak kode besar ke klien, dan overhead dalam sumber daya komputasi. Idealnya, organisasi Anda perlu mengamanatkan bagaimana dependensi dalam arsitektur frontend terdistribusi dipertahankan.

Tiga strategi yang layak untuk mengamanatkan pemeliharaan ketergantungan tidak berbagi apa-apa, menggunakan standar web seperti peta impor, dan federasi modul. Pendekatan lain adalah anti-pola karena melanggar prinsip-prinsip dasar arsitektur terdistribusi.

Bagikan apa pun, jika memungkinkan

Pendekatan share-nothing mendalilkan bahwa tidak ada dependensi antara artefak perangkat lunak independen yang harus dibagikan sama sekali, atau setidaknya tidak pada integrasi atau runtime. Ini berarti bahwa jika dua frontend mikro bergantung pada perpustakaan yang sama, masing-masing harus memanggang di perpustakaan pada waktu pembuatan dan mengirimkannya secara terpisah. Selain itu, setiap micro-frontend harus memvalidasi bahwa pustaka tidak mencemari ruang nama global dan sumber daya bersama.

Ini mengarah pada redundansi, tetapi ini adalah pertukaran sadar dengan kelincahan maksimum. Tanpa dependensi runtime yang dibagikan, tim memiliki fleksibilitas maksimum untuk mengembangkan perangkat lunak dengan cara apa pun yang mereka anggap berguna selama mereka melakukannya dalam lingkup solusi mereka dan tidak melanggar kontrak antarmuka apa pun.

Pada platform di mana frontend mikro mengikuti prinsip share-nothing, penting untuk menjaga frontend mikro seringan mungkin. Ini membutuhkan pengembang yang terampil dan rajin dalam mengoptimalkan frontend mikro mereka untuk kinerja dan yang tidak mengorbankan pengalaman pengguna untuk pengalaman pengembang.

Saat Anda membagikan kode

Saat Anda membuat keputusan untuk membagikan beberapa kode, Anda dapat membagikannya sebagai pustaka atau modul runtime. Misalnya, tim inti frontend mengirimkan pustaka untuk konsumsi mikro-frontend melalui. CDNs Tim nilai bisnis dapat memuat pustaka saat runtime, atau mereka dapat menggunakan repositori paket untuk mempublikasikan pustaka mereka. Tim micro-frontend dapat mengembangkan terhadap versi tertentu dari pustaka paket pada waktu pembuatan, mirip dengan aplikasi seluler yang menggunakan kerangka kerja hybrid.

Opsi ketiga adalah menggunakan registri paket pribadi untuk mendukung integrasi waktu pembuatan pustaka umum. Ini mengurangi risiko bahwa perubahan yang melanggar dalam kontrak pustaka memulai kesalahan saat runtime. Namun, pendekatan yang lebih konservatif ini membutuhkan lebih banyak tata kelola untuk menyinkronkan semua frontend mikro dengan versi perpustakaan yang lebih baru.

Untuk meningkatkan waktu pemuatan halaman, micro-frontend dapat mengeksternalisasi dependensi pustaka yang akan dimuat dari potongan cache dari CDN seperti Amazon. CloudFront

Untuk mengelola dependensi runtime, micro-frontend dapat menggunakan import-maps (atau pustaka seperti`System.js`) untuk menentukan dari mana setiap modul dimuat saat runtime. webpack Module Federation adalah pendekatan lain untuk menunjuk ke versi modul jarak jauh yang dihosting dan menyelesaikan dependensi umum di seluruh frontend mikro independen.

Pendekatan lain adalah memfasilitasi pemuatan dinamis peta impor dengan permintaan awal ke titik akhir penemuan.

Status bersama

Untuk mengurangi kopling mikro-frontend, penting untuk menghindari manajemen negara global yang dapat diakses dari semua frontend mikro dalam tampilan yang sama, mirip dengan arsitektur monolitik. Misalnya, memiliki toko Redux global yang dapat diakses dari semua frontend mikro meningkatkan kopling.

Pola untuk menghilangkan status bersama adalah merangkum dalam frontend mikro, dan berkomunikasi dengan pesan asinkron seperti yang dibahas sebelumnya.

Jika benar-benar diperlukan, perkenalkan antarmuka yang terdefinisi dengan baik untuk status global, dan pilih untuk berbagi hanya-baca untuk menghindari perilaku yang tidak terduga:

- Ketika ada pemisahan vertikal, Anda dapat menggunakan komponen URL dan penyimpanan browser untuk mengakses informasi dari lingkungan host.
- Bila Anda memiliki split campuran, Anda juga dapat menggunakan event atau JavaScript pustaka kustom standar DOM, seperti pemancar peristiwa atau aliran dua arah, untuk meneruskan informasi ke frontend mikro.

Jika Anda perlu berbagi beberapa informasi di seluruh frontend mikro, kami sarankan untuk meninjau kembali batas mikro-frontend. Kebutuhan untuk berbagi mungkin disebabkan oleh evolusi bisnis atau desain awal di bawah standar.

Dimungkinkan juga untuk menggunakan sesi sisi server, di mana setiap mikro-frontend mengambil data yang diperlukan dengan menggunakan pengenal sesi. Untuk mengurangi kopling, penting untuk menghilangkan status bersama dan memisahkan data sesi spesifik mikro-frontend.

Kerangka kerja dan alat

Tidak ada kekurangan kerangka kerja frontend, seperti Angular dan Next.js, tetapi kebanyakan dari mereka tidak dibuat dengan mempertimbangkan frontend mikro. Oleh karena itu, mereka terkadang kehilangan mekanisme untuk mengatasi tantangan arsitektur mikro-frontend.

Pertimbangan kerangka umum

Panduan ini tidak bertujuan untuk merekomendasikan, atau membandingkan, kerangka kerja individu. Karena beberapa frontend mikro sering berjalan di halaman aplikasi web yang sama, kinerja pemuatan dan runtime menjadi perhatian utama. Penting untuk memilih kerangka kerja yang memperkenalkan overhead sesedikit mungkin.

Kerangka kerja dibagi berdasarkan lapisan rendering:

- Rendering sisi klien (CSR)
- Rendering sisi server (SSR)

Arsitektur frontend mencakup kemampuan lain, seperti pembuatan situs statis (SSG). Namun, SSG dilakukan satu kali saja. Micro-frontend terutama disusun saat runtime, jadi CSR dan SSR adalah opsi utama.

Rendering sisi klien

Untuk CSR, ada dua opsi populer:

- Kerangka SPA tunggal
- Federasi Modul

Single SPA adalah pilihan ringan untuk menyusun frontend mikro. Ini memecahkan tantangan paling umum dalam arsitektur mikro-frontend, seperti menyusun beberapa frontend mikro di halaman yang sama dan menghindari bentrokan ketergantungan.

Federasi Modul dimulai sebagai plugin, ditawarkan oleh webpack 5, dan memecahkan sebagian besar tantangan dalam arsitektur mikro-frontend, termasuk manajemen dependensi di berbagai artefak. Module Federation 2.0 bekerja secara native dengan Rspack, webpack, esbuild, dan sekarang dengan. JavaScript

Pertimbangkan untuk tidak menggunakan kerangka kerja sama sekali. Browser modern, dengan pangsa pasar 98 persen secara keseluruhan menurut caniuse.com, menawarkan fitur seperti elemen khusus secara native, dan mereka memadai untuk aplikasi micro-frontend. Jika perlu, gabungkan elemen kustom dengan pustaka ringan untuk propagasi acara, internasionalisasi, atau masalah spesifik lainnya.

Rendering sisi server

Di sisi SSR, dua opsi utama lebih rumit:

- Rangkullah kerangka kerja yang ada seperti Next.js, dan terapkan prinsip mikro-frontend yang menggunakan Federasi Modul.
- Gunakan HTML- over-the-wire untuk bertukar fragmen HTML yang mewakili frontend mikro, dan buat fragmen ini di dalam template saat runtime. Contoh dari pendekatan ini adalah Podium.

Integrasi API - Backend untuk frontend

Pola Backends for Frontend (BFF) biasanya digunakan di lingkungan microservices. Dalam konteks micro-frontend, BFF adalah layanan sisi server yang termasuk dalam micro-frontend. Tidak semua frontend mikro perlu memiliki BFF. Namun, jika Anda menggunakan BFF, itu harus berjalan di dalam konteks terbatas yang sama dan tidak dibagikan di seluruh konteks terbatas lainnya.

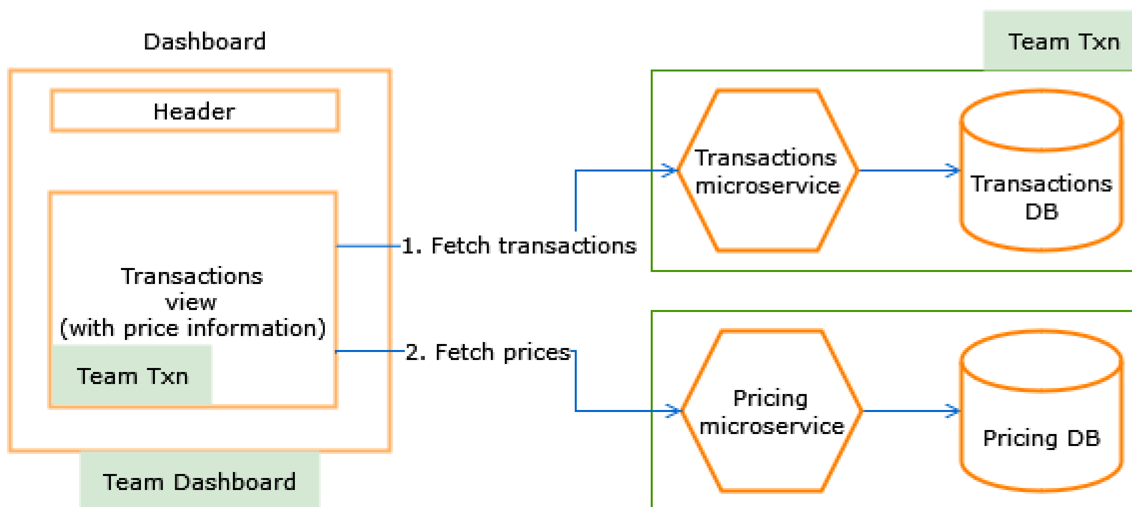
Tidak seperti layanan tradisional, BFF tidak mengikuti model domain. Sebagai gantinya, ini adalah lapisan API untuk micro-frontend untuk memproses data sebelum mencapai klien. Area di mana ini berguna termasuk yang berikut:

- Otorisasi terhadap pribadi APIs
- Agregasi data dari berbagai sumber
- Transformasi data untuk mengurangi beban jaringan dan untuk memudahkan konsumsi data oleh klien

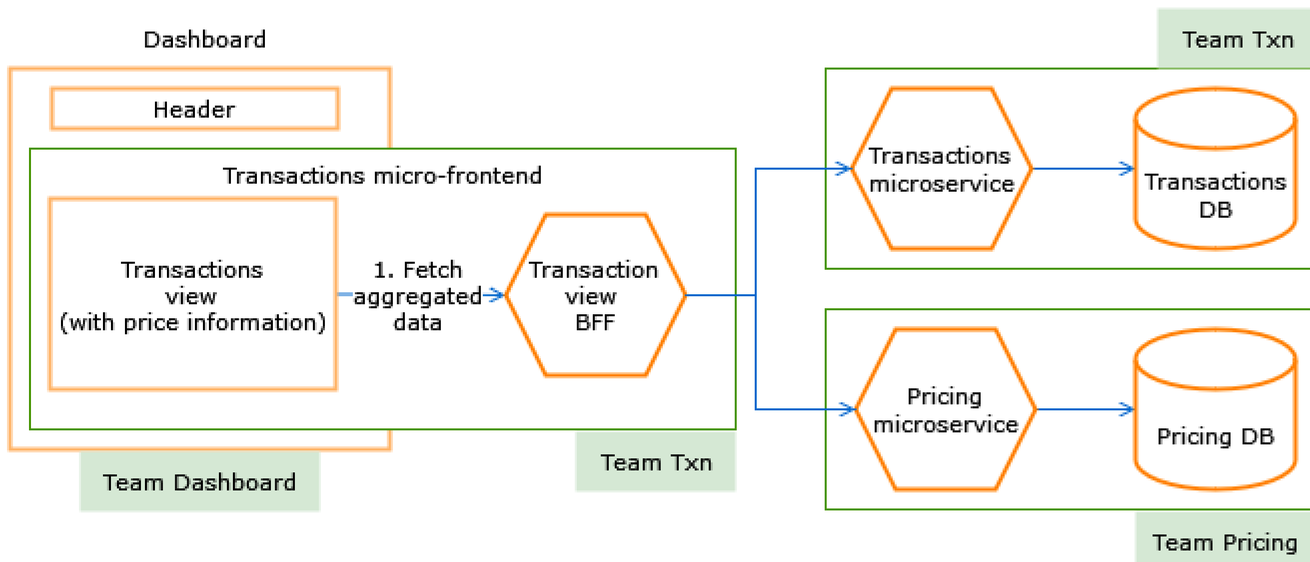
Dengan demikian, BFF dimiliki oleh micro-frontend, bukan oleh tingkat layanan domain. BFFs dapat digunakan dengan menggunakan yang berikut ini:

- AWS AppSync GraphQL APIs
- Satu set fungsi AWS Lambda
- Sebagai wadah yang berjalan di Amazon ECS, Amazon EKS, atau AWS AppRunner

Diagram berikut menunjukkan bahwa tanpa pola BFF, frontend mikro harus terhubung ke titik akhir API layanan mikro individu untuk mengambil dan mengumpulkan data.



Sebaliknya, dengan pola BFF dalam diagram berikut, mikro-frontend dapat berkomunikasi dengan backend mereka sendiri dan mengambil data agregat.



Tim dapat mengembangkan BFFs untuk saluran yang berbeda seperti seluler, web, atau tampilan tertentu, dengan persyaratan untuk mengoptimalkan interaksi backend dengan mengurangi obrolan.

Styling dan CSS

Cascading Style Sheets (CSS) adalah bahasa untuk menentukan presentasi dokumen secara terpusat alih-alih format hard-coding untuk teks dan objek. Fitur cascading bahasa dirancang untuk mengontrol prioritas antar gaya dengan menggunakan pewarisan. Saat Anda mengerjakan frontend mikro dan membuat strategi untuk mengelola dependensi, fitur cascading bahasa bisa menjadi tantangan.

Misalnya, dua mikro-frontend hidup berdampingan pada halaman yang sama, masing-masing mendefinisikan styling sendiri untuk elemen HTML. `body` Jika masing-masing mengambil file CSS sendiri dan menempelkannya ke DOM dengan menggunakan `style` tag, file CSS akan menimpa yang pertama jika keduanya memiliki definisi untuk elemen HTML umum, nama kelas atau elemen. IDs Ada berbagai strategi untuk mengatasi masalah ini, tergantung pada strategi ketergantungan yang Anda pilih untuk mengelola gaya.

Saat ini, pendekatan paling populer untuk menyeimbangkan kinerja, konsistensi, dan pengalaman pengembang terdiri dari pengembangan dan pemeliharaan sistem desain.

Sistem desain - Pendekatan berbagi sesuatu

Pendekatan ini menggunakan sistem untuk berbagi gaya bila sesuai sambil mendukung divergensi sesekali untuk menyeimbangkan konsistensi, kinerja, dan pengalaman pengembang. Sistem desain adalah kumpulan komponen yang dapat digunakan kembali, dipandu oleh standar yang jelas. Pengembangan sistem desain biasanya didorong oleh satu tim dengan masukan dan kontribusi dari banyak tim. Dalam istilah praktis, sistem desain adalah cara untuk berbagi elemen tingkat rendah yang dapat diekspor sebagai JavaScript perpustakaan. Pengembang micro-frontend dapat menggunakan perpustakaan sebagai dependensi untuk membangun antarmuka sederhana dengan menyusun sumber daya yang tersedia dan sebagai titik awal untuk membuat antarmuka baru.

Perhatikan contoh frontend mikro yang membutuhkan formulir. Pengalaman pengembang yang khas terdiri dari penggunaan komponen premade yang tersedia dalam sistem desain untuk menyusun kotak teks, tombol, daftar dropdown, dan elemen UI lainnya. Pengembang tidak perlu menulis gaya apa pun untuk komponen yang sebenarnya, hanya untuk tampilannya bersama. Sistem untuk membangun dan merilis dapat menggunakan Federasi Modul webpack atau pendekatan serupa untuk mendeklarasikan sistem desain sebagai ketergantungan eksternal, sehingga logika formulir dikemas tanpa menyertakan sistem desain.

Beberapa frontend mikro kemudian dapat melakukan hal yang sama untuk mengatasi masalah bersama. Ketika tim mengembangkan komponen baru yang dapat dibagi antara beberapa frontend mikro, komponen tersebut ditambahkan ke sistem desain setelah mencapai kematangan.

Keuntungan utama dari pendekatan sistem desain adalah tingkat konsistensi yang tinggi. Sementara micro-frontend dapat menulis gaya dan kadang-kadang mengesampingkannya dari sistem desain, hanya ada sedikit kebutuhan untuk itu. Elemen tingkat rendah utama tidak sering berubah, dan mereka menawarkan fungsionalitas dasar yang dapat diperpanjang secara default. Keuntungan lainnya adalah kinerja. Dengan strategi yang baik untuk membangun dan merilis, Anda dapat menghasilkan bundel bersama minimal yang diinstrumentasi oleh shell aplikasi. Anda dapat meningkatkan lebih jauh lagi ketika beberapa bundel spesifik mikro-frontend dimuat secara asinkron sesuai permintaan, dengan jejak minimal dalam hal bandwidth jaringan. Last but not least, pengalaman pengembang sangat ideal karena orang dapat fokus membangun antarmuka yang kaya tanpa menciptakan kembali roda (seperti menulis JavaScript dan CSS setiap kali tombol perlu ditambahkan ke halaman).

Kelemahannya adalah bahwa sistem desain dalam bentuk apa pun adalah ketergantungan, sehingga harus dipertahankan dan terkadang diperbarui. Jika beberapa frontend mikro memerlukan versi baru dari dependensi bersama, Anda dapat menggunakan salah satu dari berikut ini:

- Mekanisme orkestrasi yang kadang-kadang dapat mengambil beberapa versi ketergantungan bersama itu tanpa konflik
- Strategi bersama untuk memindahkan semua tanggungan untuk menggunakan versi baru

Misalnya, jika semua frontend mikro bergantung pada versi 3.0 dari sistem desain dan ada versi baru bernama 3.1 untuk digunakan secara bersama, Anda dapat menerapkan flag fitur untuk semua mikro-frontend untuk bermigrasi dengan risiko minimal. Untuk informasi selengkapnya, lihat bagian Bendera fitur. Kelemahan potensial lainnya adalah bahwa sistem desain biasanya mengatasi lebih dari sekadar gaya. Mereka juga mencakup JavaScript praktik dan alat. Aspek-aspek ini membutuhkan mencapai konsensus melalui debat dan kolaborasi.

Menerapkan sistem desain adalah investasi jangka panjang yang baik. Ini adalah pendekatan yang populer, dan harus dipertimbangkan oleh siapa pun yang bekerja pada arsitektur frontend yang kompleks. Biasanya membutuhkan insinyur frontend dan tim produk dan desain untuk berkolaborasi dan menentukan mekanisme untuk berinteraksi satu sama lain. Penting untuk menjadwalkan waktu untuk mencapai kondisi yang diinginkan. Penting juga untuk memiliki sponsor dari kepemimpinan sehingga orang dapat membangun sesuatu yang dapat diandalkan, terpelihara dengan baik, dan berkinerja dalam jangka panjang.

CSS yang dienkapsulasi sepenuhnya - Pendekatan berbagi apa-apa

Setiap micro-frontend menggunakan konvensi dan alat untuk mengatasi fitur cascading CSS. Contohnya adalah memastikan gaya setiap elemen selalu dikaitkan dengan nama kelas alih-alih ID elemen, dan nama kelas selalu unik. Dengan cara ini, semuanya tercakup pada frontend mikro individu, dan risiko konflik yang tidak diinginkan diminimalkan. Shell aplikasi biasanya bertugas memuat gaya micro-frontends setelah dimuat ke DOM, meskipun beberapa alat menggabungkan gaya bersama dengan menggunakan. JavaScript

Keuntungan utama dari tidak berbagi apa pun adalah berkurangnya risiko memperkenalkan konflik antara frontend mikro. Keuntungan lainnya adalah pengalaman pengembang. Setiap frontend mikro tidak berbagi apa pun dengan frontend mikro lainnya. Melepaskan dan menguji secara terpisah lebih sederhana dan lebih cepat.

Kerugian utama dari pendekatan share-nothing adalah potensi kurangnya konsistensi. Tidak ada sistem untuk menilai konsistensi. Bahkan jika menduplikasi apa yang dibagikan adalah tujuannya, itu menjadi tantangan saat menyeimbangkan kecepatan rilis dan kolaborasi. Mitigasi umum adalah menciptakan alat untuk mengukur konsistensi. Misalnya, Anda dapat membuat sistem untuk mengambil tangkapan layar otomatis dari beberapa frontend mikro yang dirender di halaman dengan browser tanpa kepala. Anda kemudian dapat meninjau tangkapan layar secara manual sebelum rilis. Namun, itu membutuhkan disiplin dan tata kelola. Untuk informasi selengkapnya, lihat bagian Menyeimbangkan otonomi dengan penyelarasan.

Bergantung pada kasus penggunaan, potensi kerugian lainnya adalah kinerja. Jika sejumlah besar gaya digunakan oleh semua frontend mikro, pelanggan harus mengunduh banyak kode duplikat. Itu akan berdampak negatif pada pengalaman pengguna.

Pendekatan share-nothing ini harus dipertimbangkan hanya untuk arsitektur micro-frontend yang hanya melibatkan beberapa tim, atau frontend mikro yang dapat mentolerir konsistensi rendah. Ini juga bisa menjadi langkah awal alami saat organisasi sedang mengerjakan sistem desain.

CSS Global Bersama - Pendekatan berbagi semua

Dengan pendekatan ini, semua kode yang terkait dengan styling disimpan dalam repositori pusat di mana kontributor menulis CSS untuk semua micro-frontend dengan bekerja pada file CSS atau dengan menggunakan preprocessors seperti Sass. Ketika perubahan dibuat, sistem build membuat

bundel CSS tunggal yang dapat di-host di CDN dan disertakan dalam setiap micro-frontend oleh shell aplikasi. Pengembang micro-frontend dapat merancang dan membangun aplikasi mereka dengan menjalankan kode mereka melalui shell aplikasi yang dihosting secara lokal.

Terlepas dari keuntungan nyata dari mengurangi risiko konflik antara frontend mikro, keuntungan dari pendekatan ini adalah konsistensi dan kinerja. Namun, memisahkan gaya dari markup dan logika membuat pengembang lebih sulit untuk memahami bagaimana gaya digunakan, bagaimana mereka dapat berkembang, dan bagaimana mereka dapat ditinggalkan. Misalnya, mungkin lebih cepat untuk memperkenalkan nama kelas baru daripada belajar tentang kelas yang ada dan konsekuensi dari mengedit propertinya. Kerugian dari membuat nama kelas baru adalah pertumbuhan ukuran bundel, yang memengaruhi kinerja, dan potensi pengenalan inkonsistensi dalam pengalaman pengguna.

Meskipun CSS global bersama dapat menjadi titik awal monolith-to-micro-frontends migrasi, jarang bermanfaat untuk arsitektur mikro-frontend yang melibatkan lebih dari satu atau dua tim yang berkolaborasi bersama. Kami merekomendasikan berinvestasi dalam sistem desain sesegera mungkin dan menerapkan pendekatan share-nothing saat sistem desain sedang dalam pengembangan.

Organisasi dan cara kerja

Seperti semua strategi arsitektur, frontend mikro memiliki implikasi jauh melampaui teknologi yang dipilih organisasi untuk diterapkan. Keputusan untuk membangun aplikasi micro-frontend harus selaras dengan bisnis, produk, organisasi, operasi, dan bahkan budaya (misalnya, memberdayakan tim dan desentralisasi pengambilan keputusan). Sebagai imbalannya, jenis arsitektur micro-frontend ini mendukung pengembangan yang benar-benar gesit dan didorong oleh produk, karena secara signifikan mengurangi overhead komunikasi antara tim independen.

Pengembangan tangkas

Gagasan pengembangan perangkat lunak tangkas telah menjadi begitu universal dalam beberapa tahun terakhir sehingga hampir setiap organisasi mengklaim bekerja gesit. Sementara definisi konklusif agile berada di luar cakupan strategi ini, ada baiknya meninjau elemen-elemen kunci yang relevan dengan pengembangan mikro-frontend.

Landasan paradigma tangkas adalah Agile Manifesto (2001), yang mendalilkan empat prinsip utama (misalnya, “Individu dan interaksi atas proses dan alat”) dan dua belas prinsip. Kerangka kerja proses seperti Scrum dan Scaled Agile Framework (SAFe) telah muncul di sekitar Manifesto Agile dan telah menemukan jalan mereka ke dalam praktik sehari-hari. Namun, filosofi di balik mereka sebagian besar disalahpahami atau diabaikan.

Dalam konteks arsitektur mikro-frontend, prinsip-prinsip tangkas berikut ini penting untuk dirangkul:

- “Sering-seringlah mengirimkan perangkat lunak yang berfungsi, dari beberapa minggu hingga beberapa bulan, dengan preferensi ke skala waktu yang lebih pendek.”

  Prinsip ini menekankan betapa pentingnya bekerja secara bertahap dan mengirimkan perangkat lunak ke produksi secara teratur dan sesering mungkin. Dari perspektif teknologi, ini mengacu pada integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD). In CI/CD, alat dan proses untuk membangun, menguji, dan penyebaran adalah bagian integral dari setiap proyek perangkat lunak. Prinsip ini juga menyiratkan bahwa infrastruktur runtime dan tanggung jawab operasional harus dimiliki oleh tim. Kepemilikan itu sangat penting dalam sistem terdistribusi di mana subsistem independen mungkin memiliki persyaratan yang berbeda secara signifikan untuk infrastruktur dan operasi.
- “Bangun proyek di sekitar individu yang termotivasi. Beri mereka lingkungan dan dukungan yang mereka butuhkan, dan percayakan mereka untuk menyelesaikan pekerjaan.”

> “Arsitektur, persyaratan, dan desain terbaik muncul dari tim yang mengatur diri sendiri.”
>
> Kedua prinsip ini menekankan manfaat kepemilikan, kemandirian, dan end-to-end tanggung jawab. Arsitektur micro-frontend akan berhasil ketika (dan hanya ketika) tim benar-benar memiliki frontend mikro mereka. End-to-endtanggung jawab dari konsepsi melalui desain dan implementasi hingga pengiriman dan operasi memastikan bahwa tim benar-benar dapat menjalankan kepemilikan. Kemandirian ini diperlukan, baik secara teknis maupun organisasi, agar tim memiliki otonomi atas arah strategis. Kami tidak menyarankan menggunakan platform mikro-frontend di organisasi terpusat yang menggunakan model pengembangan air terjun.

Komposisi dan ukuran tim

Agar tim perangkat lunak dapat menjalankan kepemilikan, ia harus mengatur dirinya sendiri, termasuk bagaimana dan apa yang diberikan tim, dalam batas-batas yang diberlakukan oleh organisasi.

Agar efektif, tim harus mampu memberikan perangkat lunak secara mandiri dan memiliki wewenang untuk memutuskan cara terbaik untuk mengirimkannya. Tim yang menerima persyaratan fitur dari manajer produk eksternal atau desain UI dari desainer eksternal, tanpa terlibat dalam perencanaan item ini, tidak dapat dianggap otonom. Fitur tersebut mungkin melanggar kontrak atau fungsionalitas yang ada. Pelanggaran semacam itu akan membutuhkan diskusi dan negosiasi lebih lanjut, dengan risiko menunda pengiriman dan memperkenalkan konflik yang tidak perlu antar tim.

Pada saat yang sama, tim tidak boleh menjadi terlalu besar. Sementara tim yang lebih besar memiliki lebih banyak sumber daya dan dapat mengakomodasi ketidakhadiran individu, kompleksitas komunikasi tumbuh secara eksponensial dengan setiap anggota baru. Tidak mungkin untuk menyatakan ukuran tim maksimum yang valid secara universal. Jumlah orang yang dibutuhkan untuk suatu proyek tergantung pada faktor-faktor seperti kematangan tim, kompleksitas teknis, laju inovasi, dan infrastruktur. Misalnya, Amazon mengikuti aturan dua pizza: Tim yang terlalu besar untuk diberi makan dua pizza harus dibagi menjadi tim yang lebih kecil. Itu bisa menjadi tantangan. Perpecahan harus terjadi di sepanjang batas-batas alami dan harus memberikan masing-masing tim otonomi dan kepemilikan atas pekerjaan mereka.

DevOps budaya

DevOps mengacu pada praktik rekayasa perangkat lunak di mana langkah-langkah siklus hidup pengembangan terintegrasi erat dari perspektif organisasi dan teknis. Berlawanan dengan

kepercayaan populer, DevOps sangat banyak tentang budaya dan pola pikir, dan sangat sedikit tentang peran dan perkakas.

Secara tradisional, organisasi perangkat lunak akan memiliki tim spesialis, seperti untuk desain, implementasi, pengujian, penyebaran, dan operasi. Setiap kali tim menyelesaikan pekerjaan mereka, mereka akan menyerahkan proyek itu ke tim berikutnya. Namun, pengiriman perangkat lunak melalui silo tim khusus menyebabkan gesekan selama serah terima. Pada saat yang sama, ketika spesialis dipaksa untuk bekerja dengan fokus yang sempit, mereka kekurangan pengetahuan di domain tetangga, dan mereka tidak memiliki pandangan sistemik tentang produk. Defisit tersebut dapat menyebabkan koherensi produk perangkat lunak yang rendah.

Misalnya, ketika seorang arsitek perangkat lunak merancang solusi yang akan diterapkan oleh seseorang di tim yang berbeda, mereka mungkin mengabaikan aspek yang melekat pada implementasi (seperti ketidakcocokan ketergantungan). Pengembang kemudian mengambil jalan pintas (seperti patch monyet), atau back-and-forth diformalkan dimulai antara arsitek dan tim pengembangan. Karena overhead mengelola proses ini, pengembangan tidak lagi gesit (dalam arti fleksibel, adaptif, inkremental, dan informal).

Meskipun istilah ini DevOps terutama berkaitan dengan budaya, ini menyiratkan teknologi dan proses yang DevOps memungkinkan dalam praktik. DevOps terkait erat denganCI/CD. When a developer has finished implementing an increment of software, they commit it to a version-control system such as Git. Traditionally, a build system would then build and integrate the software, and have it tested and released in a more or less unified and centralized process. With CI/CD, pembangunan, integrasi, pengujian, dan rilis perangkat lunak melekat dan otomatis. Idealnya, proses adalah bagian dari proyek perangkat lunak itu sendiri melalui file konfigurasi yang disesuaikan dengan proyek yang diberikan secara khusus.

Sebanyak mungkin langkah otomatis. Misalnya, praktik pengujian manual harus dikurangi, karena hampir semua jenis pengujian dapat diotomatisasi. Ketika proyek diatur dengan cara itu, pembaruan untuk produk perangkat lunak dapat dikirimkan beberapa kali setiap hari dengan keyakinan tinggi. Teknologi lain yang mendukung DevOps adalah infrastruktur sebagai kode (IAc).

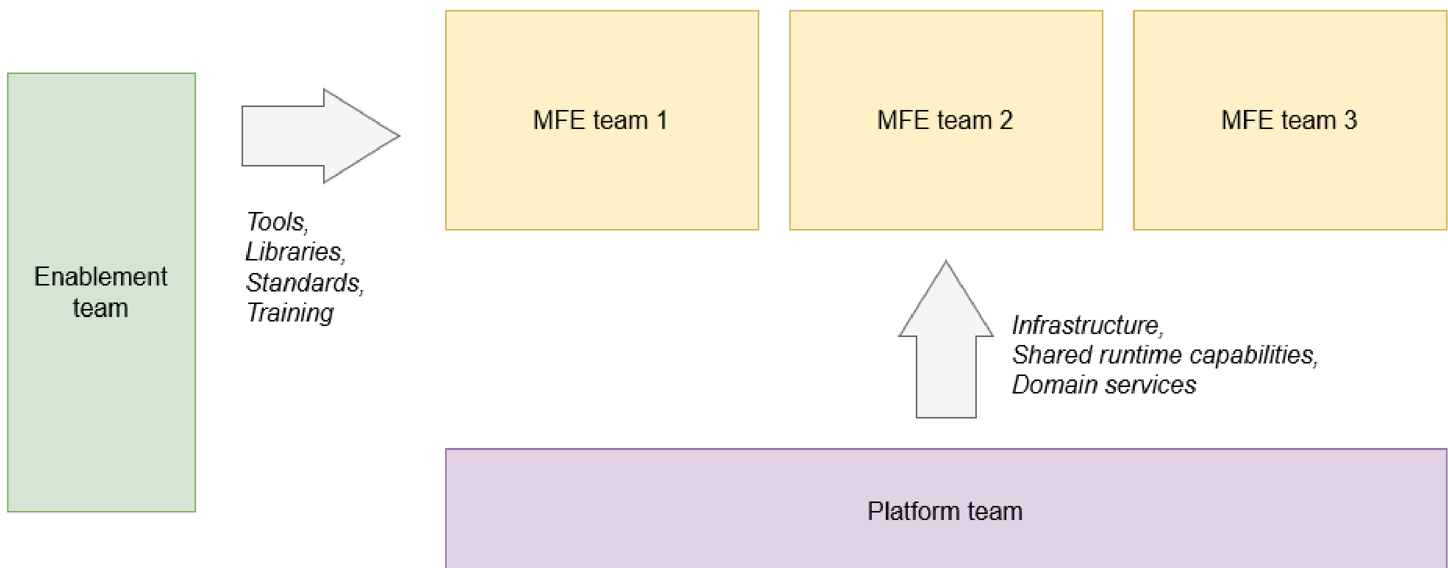
Secara tradisional, menyiapkan dan memelihara infrastruktur TI akan membutuhkan pekerjaan manual untuk menginstal dan memelihara perangkat keras (menyiapkan kabel dan server di pusat data) dan perangkat lunak operasional. Ini perlu, tetapi memiliki banyak kelemahan. Pengaturan memakan waktu dan rawan kesalahan. Perangkat keras sering disediakan secara berlebihan atau kurang disediakan, yang menyebabkan pengeluaran berlebih atau kinerja yang menurun. Dengan menggunakan IAC, Anda dapat menjelaskan persyaratan infrastruktur untuk sistem TI melalui file konfigurasi dari mana layanan cloud dapat digunakan dan diperbarui secara otomatis.

Apa hubungannya semua ini dengan frontend mikro? DevOps, CI/CD, dan IAc adalah pelengkap ideal untuk arsitektur micro-frontend. Manfaat mikro-frontend bergantung pada proses pengiriman yang cepat dan tanpa gesekan. DevOps Budaya dapat berkembang hanya di lingkungan di mana tim memiliki proyek perangkat lunak dengan end-to-end tanggung jawab.

Mengatur pengembangan mikro-frontend di beberapa tim

Saat menskalakan pengembangan mikro-frontend di beberapa tim lintas fungsi, dua masalah muncul: Pertama, tim mulai mengembangkan interpretasi paradigma mereka sendiri, membuat pilihan kerangka kerja dan perpustakaan, dan membuat perpustakaan perkakas dan pembantu mereka sendiri. Kedua, tim yang sepenuhnya otonom harus bertanggung jawab atas kemampuan generik seperti manajemen infrastruktur tingkat rendah. Oleh karena itu, masuk akal untuk memperkenalkan dua tim tambahan dalam organisasi mikro-frontend multi-tim: tim pemberdayaan dan tim platform. Konsep-konsep ini diadopsi secara luas dalam organisasi TI modern dengan sistem terdistribusi dan didokumentasikan dengan baik dalam Topologi Tim.

Diagram berikut menunjukkan tim pemberdayaan yang menyediakan alat, pustaka, standar, dan pengujian untuk tiga tim mikro-frontend. Tim platform menyediakan infrastruktur, kemampuan runtime bersama, dan layanan domain untuk tiga tim micro-frontend yang sama.



Tim platform mendukung tim mikro-frontend dengan membebaskan mereka dari angkat berat yang tidak berdiferensiasi. Dukungan ini dapat mencakup layanan infrastruktur seperti runtime kontainer, CI/CD pipeline, alat kolaborasi, dan pemantauan. Namun, menyiapkan tim platform tidak boleh mengarah ke organisasi di mana pengembangan terlepas dari operasi. Yang sebaliknya adalah

benar: Tim platform menawarkan produk teknik, dan tim micro-frontend memiliki kepemilikan dan tanggung jawab runtime atas layanan mereka di platform.

Tim pemberdayaan memberikan dukungan dengan berfokus pada tata kelola dan memastikan konsistensi di seluruh tim mikro-frontend. (Tim platform tidak boleh terlibat dengan ini.) Tim pemberdayaan mengelola sumber daya bersama seperti pustaka UI, dan menciptakan standar seperti pilihan kerangka kerja, anggaran kinerja, dan konvensi interoperabilitas. Pada saat yang sama, ini memberikan pelatihan kepada tim atau anggota tim baru dalam menerapkan standar dan perkakas sebagaimana didefinisikan oleh tata kelola.

Men-deploy

Bintang utara untuk otonomi dalam tim mikro-frontend adalah memiliki pipa otomatis dengan jalur produksi yang independen dari tim mikro-frontend lainnya. Tim yang mengikuti prinsip share-nothing dapat menerapkan jaringan pipa independen. Tim yang berbagi pustaka atau bergantung pada tim platform harus memutuskan cara mengelola dependensi dalam pipeline penerapan.

Biasanya, setiap pipa melakukan hal berikut:

- Membangun aset frontend
- Menyebarkan aset ke hosting untuk konsumsi
- Memastikan bahwa registri dan cache diperbarui sehingga versi baru dapat dikirimkan ke pelanggan

Langkah-langkah pipeline sebenarnya bervariasi tergantung pada tumpukan teknologi dan pendekatan komposisi halaman.

Untuk komposisi sisi klien, ini berarti mengunggah bundel aplikasi ke bucket hosting, dan melepaskan ke konsumsi melalui caching di CDN. Aplikasi yang menggunakan cache browser dengan pekerja layanan juga harus menerapkan cara untuk memperbarui cache pekerja layanan.

Untuk komposisi sisi server, ini biasanya berarti menyebarkan versi baru komponen server dan memperbarui registri mikro-frontend untuk membuat versi baru dapat ditemukan. Anda dapat menggunakan blue/green atau pola penyebaran kenari untuk secara bertahap meluncurkan versi baru.

Tata kelola

Beberapa persona biasanya bekerja pada frontend mikro, dan masing-masing bekerja di bawah kendala yang berbeda menuju tujuan bisnis bersama. Sementara komunikasi dan kolaborasi antara orang-orang adalah kunci keberhasilan, komunikasi yang berlebihan dan menerapkan proses yang terlalu rumit memperlambat siklus pengembangan. Hal ini mengakibatkan penurunan moral, dan menurunkan kualitas bar.

Perusahaan paling sukses yang menerapkan frontend mikro dengan menggunakan banyak tim menciptakan mekanisme untuk menyeimbangkan otonomi dengan keselarasan. Mereka memberdayakan para pengambil keputusan untuk mengambil tindakan secara lokal dan meningkat secara hierarkis hanya bila diperlukan. Mekanisme meliputi:

- Kontrak API
- Interaktivitas silang menggunakan acara
- Menyeimbangkan otonomi dengan keselarasan
- Bendera fitur
- Penemuan layanan

Kontrak API

Setiap micro-frontend adalah sistem yang mampu merangkum opini, logika, dan kompleksitas. Kekhawatiran lintas sektoral biasanya meliputi yang berikut:

- Sistem desain - Perkakas untuk mengembangkan UIs didistribusikan sebagai perpustakaan
- Komposisi - Cara micro-frontend perlu berinteraksi dengan shell aplikasi untuk merender dan mewarisi konteksnya
- Penanganan logika - Interaksi dengan APIs untuk menangani keadaan persisten
- Interaktivitas dengan frontend mikro lainnya - Skenario seperti menerbitkan dan mengkonsumsi acara atau menavigasi dari satu mikro-frontend ke yang lain

Untuk mempercepat konsumsi dan pemecahan masalah, biasanya berinvestasi dalam standarisasi cara antarmuka ini dideklarasikan dan didokumentasikan, termasuk dependensi mikro-frontend. Wiki yang dikuratori oleh manusia adalah awal yang baik. Pendekatan yang lebih skalabel

adalah menyimpan informasi ini sebagai metadata terstruktur dalam kode. Anda kemudian dapat memusatkannya untuk konsumsi dengan menggunakan otomatisasi untuk melacak perubahan historis dan menyediakan pencarian teks lengkap.

Ketika mikro-frontend melibatkan sejumlah besar tim, Anda memerlukan strategi untuk berkoordinasi antar tim. Berbagi kontrak API secara terpadu menjadi suatu keharusan karena mengurangi overhead komunikasi dan meningkatkan pengalaman pengembang.

OpenAPI adalah bahasa spesifikasi untuk HTTP APIs yang mendukung mendefinisikan antarmuka dan kontrak API secara terpadu. Anda dapat menerapkan REST APIs dengan menggunakan OpenAPI di Amazon API Gateway. Anda juga dapat menggunakan berbagai kerangka kerja open source yang dapat Anda host dalam wadah atau mesin virtual. Keuntungan yang signifikan adalah OpenAPI dapat secara otomatis menghasilkan dokumentasi dalam format yang konsisten, sehingga banyak tim dapat berbagi pengetahuan dengan investasi awal yang minimal.

Ketika beberapa tim bekerja pada frontend mikro, mereka sering membentuk kelompok. Dalam kelompok-kelompok ini, orang dapat bertemu dan belajar dari satu sama lain sambil memikirkan dan berkontribusi pada gambaran yang lebih besar. Inisiatif ini biasanya mendefinisikan dan mendokumentasikan batas-batas kepemilikan, mendiskusikan masalah lintas sektoral, dan mengidentifikasi sejak dini setiap duplikasi upaya untuk memecahkan masalah umum.

Interaktivitas silang menggunakan acara

Dalam beberapa skenario, beberapa frontend mikro mungkin perlu berinteraksi satu sama lain untuk bereaksi terhadap perubahan status atau tindakan pengguna. Misalnya, beberapa frontend mikro pada halaman dapat menyertakan menu yang dapat dilipat. Menu muncul ketika pengguna memilih tombol. Menu disembunyikan ketika pengguna mengklik di tempat lain, termasuk menu lain yang dirender dalam micro-frontend yang berbeda.

Secara teknis, pustaka negara bersama seperti Redux dapat digunakan oleh beberapa frontend mikro dan dikoordinasikan oleh shell. Namun, itu menciptakan kopling yang signifikan antara aplikasi, menghasilkan kode yang lebih sulit untuk diuji, dan mungkin memperlambat kinerja selama rendering.

Salah satu pendekatan yang umum dan efektif adalah mengembangkan bus acara yang didistribusikan sebagai perpustakaan, diatur oleh shell aplikasi, dan digunakan oleh beberapa mikro-frontend. Dengan cara ini, setiap mikro-frontend menerbitkan dan mendengarkan peristiwa tertentu secara asinkron, mendasarkan perilakunya pada keadaan internalnya sendiri saja. Kemudian,

beberapa tim dapat memelihara halaman wiki bersama yang menjelaskan peristiwa dan perilaku dokumen yang telah disepakati oleh desainer pengalaman pengguna.

Dalam implementasi contoh event-bus, komponen dropdown menggunakan bus bersama untuk mempublikasikan acara yang dipanggil `drop-down-open-menu` dengan muatan. `{"id": "homepage-aboutus-button"}` Komponen menambahkan pendengar ke `drop-down-open-menu` acara untuk memastikan bahwa jika suatu peristiwa diaktifkan untuk ID baru, komponen dropdown dirender untuk menyembunyikan bagian yang dapat dilipat. Dengan cara ini, mikro-frontend dapat bereaksi terhadap perubahan secara asinkron dengan peningkatan kinerja dan enkapsulasi yang lebih baik, memudahkan banyak tim untuk merancang dan menguji perilaku.

Kami merekomendasikan penggunaan standar yang APIs diterapkan secara native oleh browser modern untuk meningkatkan kesederhanaan dan pemeliharaan. Referensi MDN Event memberikan informasi tentang penggunaan peristiwa dengan aplikasi yang diberikan sisi klien.

Menyeimbangkan otonomi dengan keselarasan

Arsitektur mikro-frontend sangat bias terhadap otonomi tim. Namun, penting untuk membedakan antara area yang dapat mendukung fleksibilitas dan beragam pendekatan untuk memecahkan masalah, dan area di mana standardisasi diperlukan untuk mencapai keselarasan. Pemimpin dan arsitek senior harus mengidentifikasi area ini sejak dini dan memprioritaskan investasi untuk menyeimbangkan keamanan, kinerja, keunggulan operasional, dan keandalan frontend mikro. Menemukan keseimbangan ini melibatkan hal-hal berikut: pembuatan mikro-frontend, pengujian, rilis, dan pencatatan, pemantauan, dan peringatan.

Membuat frontend mikro

Idealnya, semua tim sangat selaras untuk memaksimalkan manfaat dalam hal kinerja pengguna akhir. Dalam praktiknya, ini bisa sulit, dan mungkin membutuhkan lebih banyak usaha. Kami merekomendasikan memulai dengan beberapa pedoman tertulis yang dapat disumbangkan oleh banyak tim melalui debat terbuka dan transparan. Tim kemudian dapat secara bertahap mengadopsi pola perangkat lunak Cookiecutter, yang mendukung pembuatan alat yang menyediakan cara terpadu untuk perancah proyek.

Dengan menggunakan pendekatan ini, Anda dapat memanggang pendapat dan kendala. Kelemahannya adalah bahwa alat ini memerlukan investasi yang signifikan untuk pembuatan dan pemeliharaan, dan untuk memastikan bahwa pemblokir ditangani dengan cepat tanpa mempengaruhi produktivitas pengembang.

End-to-end pengujian untuk mikro-frontend

Pengujian unit dapat diserahkan kepada pemilik. Kami merekomendasikan untuk menerapkan strategi sejak dini untuk melakukan cross-test micro-frontend yang berjalan pada shell unik. Strategi ini mencakup kemampuan untuk menguji aplikasi sebelum dan sesudah rilis produksi. Kami merekomendasikan pengembangan proses dan dokumentasi untuk orang-orang teknis dan nonteknis untuk menguji fungsionalitas kritis secara manual.

Penting untuk memastikan bahwa perubahan tidak menurunkan pengalaman pelanggan fungsional atau nonfungsional. Strategi yang ideal adalah berinvestasi secara bertahap dalam pengujian otomatis, baik untuk fitur utama maupun untuk karakteristik arsitektur seperti keamanan dan kinerja.

Melepaskan mikro-frontend

Setiap tim mungkin memiliki caranya sendiri untuk menyebarkan kode mereka, membuat opini, dan memiliki infrastruktur. Biaya kompleksitas untuk memelihara sistem seperti itu biasanya merupakan pencegah. Sebagai gantinya, kami merekomendasikan investasi sejak dini untuk menerapkan strategi bersama yang dapat ditegakkan oleh alat bersama.

Kembangkan template dengan platform CI/CD pilihan. Tim kemudian dapat menggunakan templat yang telah disetujui sebelumnya dan infrastruktur bersama untuk merilis perubahan pada produksi. Anda dapat mulai berinvestasi dalam pekerjaan pengembangan ini lebih awal karena sistem ini jarang memerlukan pembaruan yang signifikan setelah periode awal pengujian dan konsolidasi.

Pencatatan log dan pemantauan

Setiap tim dapat memiliki metrik bisnis dan sistem yang berbeda yang ingin mereka lacak untuk tujuan operasional atau analitik. Pola perangkat lunak Cookiecutter dapat diterapkan di sini juga. Pengiriman acara dapat diabstraksikan dan tersedia sebagai pustaka yang dapat dikonsumsi oleh beberapa frontend mikro. Untuk menyeimbangkan fleksibilitas dan memberikan otonomi, kembangkan alat untuk mencatat metrik kustom dan membuat dasbor atau laporan khusus. Pelaporan mempromosikan kolaborasi erat dengan pemilik produk dan mengurangi umpan balik pelanggan akhir.

Dengan menstandarisasi pengiriman, beberapa tim dapat berkolaborasi untuk melacak metrik. Misalnya, situs web e-commerce dapat melacak perjalanan pengguna dari frontend mikro “Detail produk” ke frontend mikro “Keranjang”, ke frontend mikro “Pembelian” untuk mengukur keterlibatan, churn, dan masalah. Jika setiap micro-frontend mencatat peristiwa dengan menggunakan satu

pustaka, Anda dapat menggunakan data ini secara keseluruhan, menjelajahinya secara holistik, dan mengidentifikasi tren yang berwawasan luas.

Peringatan

Mirip dengan penebangan dan pemantauan, mengingatkan manfaat dari standardisasi dengan ruang untuk tingkat fleksibilitas. Tim yang berbeda mungkin bereaksi berbeda terhadap peringatan fungsional dan non-fungsional. Namun, jika semua tim memiliki cara konsolidasi untuk memulai peringatan berdasarkan metrik yang dikumpulkan dan dianalisis pada platform bersama, bisnis dapat mengidentifikasi masalah lintas tim. Kemampuan ini berguna selama peristiwa manajemen insiden. Misalnya, peringatan dapat dimulai dengan yang berikut:

- Peningkatan jumlah pengecualian JavaScript sisi klien pada versi browser tertentu
- Waktu untuk membuat terdegradasi secara signifikan di atas ambang batas tertentu
- Peningkatan jumlah kode status 5xx saat menggunakan API tertentu

Bergantung pada kematangan sistem Anda, Anda dapat menyeimbangkan upaya Anda di berbagai bagian infrastruktur Anda, seperti yang ditunjukkan pada tabel berikut.

Adopsi	Penelitian dan pengembangan	Pendakian	Kedewasaan
Buat frontend mikro.	Eksperimen, dokumentasikan, dan bagikan pembelajaran.	Investasikan dalam perkakas untuk perancah mikro-frontend baru. Menginjili adopsi.	Konsolidasikan perkakas untuk perancah. Dorong untuk adopsi.
Uji mikro-fro ntend ujung ke ujung.	Menerapkan mekanisme untuk menguji secara manual semua frontend mikro terkait.	Investasikan dalam perkakas untuk pengujian keamanan dan kinerja otomatis. Selidiki bendera fitur dan penemuan layanan.	Mengkonsolidasikan perkakas untuk penemuan layanan, pengujian dalam produksi, dan pengujian otomatis end-to-end.

Adopsi	Penelitian dan pengembangan	Pendakian	Kedewasaan
Lepaskan mikro-fro ntend.	Investasikan dalam infrastru ktur CI/CD bersama dan rilis multi-lingkungan otomatis. Menginjili adopsi.	Konsolidasi perkakas untuk infrastruktur CI/CD Menerapkan mekanisme rollback manual. Dorong untuk adopsi.	Buat mekanisme untuk memulai rollback otomatis di atas metrik dan peringatan sistem dan bisnis.
Amati kinerja mikro-fro ntend.	Berinvest asi dalam infrastruktur dan perpustakaan pemantauan bersama untuk pencatatan sistem dan acara bisnis yang konsisten.	Konsolidasikan perkakas untuk pemantauan dan peringatan. Menerapka n dasbor lintas tim untuk memantau kesehatan umum dan meningkatkan manajemen insiden.	Standarisasi skema penebanga n. Optimalkan biaya. Menerapkan peringatan berdasarkan metrik bisnis yang kompleks.

Bendera fitur

Bendera fitur dapat diimplementasikan di frontend mikro untuk memfasilitasi koordinasi pengujian dan pelepasan fitur di berbagai lingkungan. Teknik flag fitur terdiri dari memusatkan keputusan di toko berbasis Boolean, dan perilaku mengemudi berdasarkan itu. Ini sering digunakan untuk menyebarkan perubahan secara diam-diam yang dapat disembunyikan sampai saat tertentu, sambil membuka rilis baru untuk fitur baru yang seharusnya diblokir, mengurangi kecepatan tim.

Pertimbangkan contoh tim yang mengerjakan fitur micro-frontend yang akan diluncurkan pada tanggal tertentu. Fitur ini sudah siap, tetapi perlu dirilis bersama dengan perubahan pada frontend mikro lain yang dirilis secara independen. Memblokir pelepasan kedua frontend mikro akan dianggap sebagai anti-pola dan akan meningkatkan risiko saat digunakan.

Sebagai gantinya, tim dapat membuat flag fitur Boolean dalam database yang mereka berdua gunakan selama waktu render (mungkin melalui panggilan HTTP ke API Feature Flags bersama).

Tim bahkan dapat merilis perubahan dalam lingkungan pengujian di mana nilai Boolean diatur `True` untuk memverifikasi persyaratan fungsional dan nonfungsional lintas proyek sebelum diluncurkan ke produksi.

Contoh lain dari penggunaan flag fitur adalah menerapkan mekanisme untuk mengganti nilai flag dengan menetapkan nilai tertentu melalui `QueryString` parameter atau menyimpan string pengujian tertentu dalam cookie. Pemilik produk dapat mengulangi fitur tanpa memblokir rilis fitur lain atau perbaikan bug hingga tanggal peluncuran. Pada tanggal tertentu, mengubah nilai flag pada database secara instan membuat perubahan terlihat dalam produksi, tanpa perlu rilis terkoordinasi lintas tim. Setelah fitur dirilis, tim pengembangan membersihkan kode untuk menghapus perilaku lama.

Kasus penggunaan lainnya termasuk merilis sistem flag fitur berbasis konteks. Misalnya, jika satu situs web melayani pelanggan dalam berbagai bahasa, fitur mungkin hanya tersedia untuk pengunjung negara tertentu. Sistem flag fitur dapat bergantung pada konsumen yang mengirim konteks negara (misalnya dengan menggunakan header `Accept-Language` HTTP), dan mungkin ada perilaku yang berbeda tergantung pada konteks itu.

Sementara flag fitur adalah alat yang ampuh untuk memfasilitasi kolaborasi antara pengembang dan pemilik produk, mereka mengandalkan ketekunan orang untuk menghindari degradasi yang signifikan dari basis kode. Menjaga flag tetap aktif pada beberapa fitur dapat meningkatkan kompleksitas saat memecahkan masalah, meningkatkan ukuran JavaScript bundel, dan pada akhirnya mengakumulasi utang teknis. Kegiatan mitigasi umum meliputi:

- Unit menguji setiap fitur di belakang bendera untuk mengurangi kemungkinan bug, yang dapat memperkenalkan loop umpan balik yang lebih lama di pipeline CI/CD otomatis yang menjalankan pengujian
- Membuat alat untuk mengukur ukuran bundel meningkat selama perubahan kode, yang dapat dikurangi selama tinjauan kode

AWS menawarkan berbagai solusi untuk mengoptimalkan pengujian A/B di edge dengan menggunakan CloudFront fungsi Amazon atau Lambda @Edge. Pendekatan ini membantu mengurangi kompleksitas mengintegrasikan solusi atau produk SaaS yang ada yang Anda gunakan untuk menegaskan asumsi Anda. Untuk informasi lebih lanjut, lihat pengujian A/B.

Penemuan Layanan

Pola penemuan frontend meningkatkan pengalaman pengembangan saat mengembangkan, menguji, dan memberikan frontend mikro. Pola menggunakan konfigurasi yang dapat dibagikan

yang menggambarkan titik masuk mikro-frontend. Konfigurasi yang dapat dibagikan juga mencakup metadata tambahan yang digunakan untuk penerapan yang aman di setiap lingkungan dengan menggunakan rilis kenari.

Pengembangan frontend modern memerlukan penggunaan berbagai alat dan perpustakaan untuk mendukung modularitas selama pengembangan. Secara tradisional, proses ini terdiri dari bundling kode ke dalam file individual yang dapat di-host di CDN dengan tujuan menjaga panggilan jaringan minimal selama runtime, termasuk pemuatan awal (ketika aplikasi terbuka di browser) dan penggunaan (ketika pelanggan melakukan tindakan seperti memilih tombol atau memasukkan informasi).

Memisahkan bundel

Arsitektur micro-frontend memecahkan masalah kinerja yang disebabkan oleh bundel yang sangat besar yang dihasilkan dengan menggabungkan satu set fungsi yang besar secara individual. Misalnya, situs web e-commerce yang sangat besar dapat dibundel menjadi JavaScript file 6 MB. Meskipun kompresi, ukuran file itu mungkin berdampak negatif pada pengalaman pengguna saat memuat aplikasi dan mengunduh file dari CDN yang dioptimalkan tepi.

Jika Anda membagi aplikasi menjadi halaman beranda, detail produk, dan keranjang micro-frontend, Anda dapat menggunakan mekanisme bundling untuk menghasilkan tiga bundel 2 MB individual. Perubahan ini dapat meningkatkan kinerja untuk pemuatan pertama sebesar 300 persen saat pengguna menggunakan halaman beranda. Paket micro-frontend produk atau keranjang dimuat secara asinkron hanya jika dan ketika pengguna mengunjungi halaman produk untuk item dan memutuskan untuk membelinya.

Banyak kerangka kerja dan pustaka tersedia berdasarkan pendekatan ini, dan ada keuntungan bagi pelanggan dan pengembang. Untuk mengidentifikasi batas-batas bisnis yang dapat mengakibatkan decoupling dependensi dalam kode, Anda dapat memetakan fungsi bisnis yang berbeda ke beberapa tim. Kepemilikan yang didistribusikan memperkenalkan kemandirian dan kelincahan.

Saat Anda membagi paket build, Anda dapat menggunakan konfigurasi untuk memetakan mikro-frontend dan mendorong orkestrasi untuk pemuatan awal dan navigasi pasca-muat. Kemudian, konfigurasi dapat digunakan selama runtime daripada selama waktu pembuatan. Misalnya, kode frontend sisi klien atau kode backend sisi server dapat melakukan panggilan jaringan awal ke API untuk mengambil daftar frontend mikro secara dinamis. Ini juga mengambil metadata yang diperlukan untuk komposisi dan integrasi. Anda dapat mengonfigurasi strategi failover dan caching untuk keandalan dan kinerja. Pemetaan frontend mikro membantu membuat penerapan individual dari

frontend mikro dapat ditemukan oleh frontend mikro yang diterapkan sebelumnya yang diatur oleh aplikasi shell.

Rilis kenari

Rilis kenari adalah pola yang mapan dan populer untuk menyebarkan layanan mikro. Rilis Canary memasukkan pengguna target rilis ke dalam beberapa grup, dan merilis perubahan secara bertahap sebagai lawan dari penggantian langsung (juga dikenal sebagai penerapan biru/hijau). Contoh strategi rilis kenari adalah meluncurkan perubahan baru ke 10 persen dari target pengguna, dan menambahkan 10 persen setiap menit, dengan total durasi 10 menit untuk mencapai 100 persen.

Tujuan dari rilis kenari adalah untuk mendapatkan umpan balik awal tentang perubahan, memantau sistem untuk mengurangi dampak dari masalah apa pun. Ketika otomatisasi dilakukan, metrik bisnis atau sistem dapat dipantau oleh sistem internal yang dapat menghentikan penyebaran atau memulai rollback.

Misalnya, perubahan mungkin menimbulkan bug yang, dalam beberapa menit pertama rilis, mengakibatkan hilangnya pendapatan atau penurunan kinerja. Pemantauan otomatis dapat memulai alarm. Dengan pola penemuan layanan, alarm itu dapat menghentikan penyebaran dan segera memutar kembali, hanya memengaruhi 20 persen pengguna, bukan 100 persen. Bisnis mendapat manfaat dari berkurangnya ruang lingkup masalah.

Untuk contoh arsitektur yang menggunakan DynamoDB sebagai penyimpanan untuk mengimplementasikan REST Admin API, lihat solusi Frontend Service Discovery on AWS. GitHub Gunakan AWS CloudFormation template untuk mengintegrasikan arsitektur dalam pipeline CI/CD Anda sendiri. Solusinya mencakup REST Consumer API untuk mengintegrasikan solusi dengan aplikasi frontend Anda.

Apakah Anda memerlukan tim platform?

Beberapa perusahaan memiliki tim yang bertanggung jawab untuk memiliki dan memelihara kode, infrastruktur, dan proses yang diadopsi oleh tim lain untuk bekerja pada frontend mikro. Tanggung jawab umum meliputi:

- Membuat dan memelihara pipa CI/CD yang dapat digunakan dengan repositori yang berisi micro-frontend. Buat dan uji perubahan kode, dan lepaskan di berbagai lingkungan.
- Buat dan pertahankan alat terkait observabilitas seperti dasbor bersama, mekanisme peringatan, dan sistem untuk bereaksi terhadap masalah.
- Membuat dan memelihara pustaka bersama untuk penanganan peristiwa, konsumsi layanan bersama, dan dependensi pihak ketiga.
- Membuat dan memelihara alat yang terus memantau kualitas nonfungsional seperti kinerja, keamanan, dan keandalan sistem.
- Membuat dan memelihara sistem desain.
- Membuat, memelihara, dan mendukung shell aplikasi untuk sistem micro-frontend.

Bergantung pada skala proyek, Anda dapat mengelola tanggung jawab ini dengan menggunakan salah satu pendekatan berikut:

- Buat tim platform khusus yang satu-satunya tanggung jawabnya adalah bekerja pada alat bersama.
- Buat grup yang terdiri dari anggota dari beberapa tim. Anggota kelompok membagi waktu mereka antara mengerjakan frontend mikro dan mengerjakan perkakas bersama. Ini juga dikenal sebagai tim harimau.

Sementara pendekatan tim harimau adalah cara yang efektif untuk tetap fokus pada pelanggan, tim harimau sering berevolusi menjadi tim platform jika proyek mendapatkan daya tarik dan tanggung jawab. Untuk tim platform dan tim harimau, perusahaan paling sukses yang bekerja di frontend mikro membentuk tim ini sehingga banyak orang dengan berbagai latar belakang dan keterampilan dapat berkontribusi. Anggota tim mungkin termasuk insinyur backend, insinyur frontend, desainer pengalaman pengguna (UX), dan manajer produk teknis. Keragaman ini mendorong orang untuk terus terlibat dalam perdebatan dan desain yang sehat dengan mempertimbangkan kesederhanaan.

Langkah selanjutnya

Panduan ini mencakup pola arsitektur dan organisasi, trade-off untuk keputusan kunci, dan masalah tata kelola yang terkait dengan frontend mikro. Tabel merangkum pertukaran praktik yang dibahas dalam dokumen ini dalam hal dimensi berikut:

- Otonomi - Kemampuan setiap tim mikro-frontend untuk secara independen mengembangkan implementasi dan rilis mereka ke pengguna akhir.
- Konsistensi - Pengalaman keseluruhan aplikasi di mana setiap mikro-frontend berperilaku seperti yang diharapkan. Konsistensi tinggi berarti micro-frontend konsisten dengan sisa aplikasi dan tidak merugikan pengalaman pengguna dari keseluruhan aplikasi.
- Kompleksitas — Jumlah infrastruktur, kode, dan upaya yang diperlukan untuk mengimplementasikan dan menguji frontend mikro, keseluruhan aplikasi, dan kontrol tata kelola.

Berlatih	Otonomi	Konsistensi	Kompleksitas
Memban dengan frontend mikro alih-alih aplikasi monolitik	Tinggi	Sedang	Tinggi

Praktik berbagi kode	Otonomi	Konsistensi	Kompleksitas
Bagikan apa pun	Tinggi	Rendah	Rendah

Praktik berbagi kode	Otonomi	Konsistensi	Kompleksitas
Bagikan kekhawa ran lintas sektoral	Sedang	Tinggi	Sedang
Bagikan logika bisnis	Rendah	Tinggi	Sedang
Bagikan melalui perpusta aan pada waktu pembuat	Sedang	Tinggi	Rendah
Bagikan saat runtime	Tinggi	Tinggi	Tinggi

Praktik penemua mikro-frontend	Otonomi	Konsistensi	Kompleksitas
Konfigura sikan selama	Rendah	Tinggi	Rendah

Praktik penemua mikro-fro ntend	Otonomi	Konsistensi	Kompleksitas
pembuat aplikasi			
Penemu sisi server	Tinggi	Tinggi	Sedang
Penemu sisi klien (runtime)	Tinggi	Tinggi	Sedang

Lihat praktik komposi	Otonomi	Konsistensi	Kompleksitas
Komposi di sisi server	Tinggi	Sedang	Tinggi
Komposi sisi tepi	Sedang	Sedang	Tinggi
Komposi di sisi klien	Tinggi	Sedang	Sedang

Untuk mempelajari lebih lanjut tentang konsep yang diperkenalkan dalam panduan ini, lihat bagian [Sumber Daya](#).

Sumber daya

- Mikro-frontend dalam konteks
- Desain Berbasis Domain
- Visual EDA
- Penemuan Frontend
- Penemuan Layanan Frontend di AWS
- Manifesto Agile
- MDN Referensi acara
- OpenAPI

Kontributor

Individu-individu berikut berkontribusi pada panduan ini.

- Matteo Figus, Arsitek Solusi Utama, AWS
- Alexander Guensche, Arsitek Solusi Senior, AWS
- Harun Hasdal, Arsitek Solusi Senior, AWS
- Luca Mezzalira, Principal Go to Market Specialist Solutions Architect Serverless UK, AWS

Riwayat dokumen

Tabel berikut menjelaskan perubahan signifikan pada panduan ini. Jika Anda ingin diberi tahu tentang pembaruan masa depan, Anda dapat berlangganan umpan RSS.

Perubahan	Deskripsi	Tanggal
Publikasi awal	—	Juli 12, 2024

AWS Glosarium Panduan Preskriptif

Berikut ini adalah istilah yang umum digunakan dalam strategi, panduan, dan pola yang disediakan oleh Panduan AWS Preskriptif. Untuk menyarankan entri, silakan gunakan tautan Berikan umpan balik di akhir glosarium.

Nomor

7 Rs

Tujuh strategi migrasi umum untuk memindahkan aplikasi ke cloud. Strategi ini dibangun di atas 5 Rs yang diidentifikasi Gartner pada tahun 2011 dan terdiri dari yang berikut:

- Refactor/Re-Architect — Memindahkan aplikasi dan memodifikasi arsitekturnya dengan memanfaatkan sepenuhnya fitur cloud-native untuk meningkatkan kelincahan, kinerja, dan skalabilitas. Ini biasanya melibatkan porting sistem operasi dan database. Contoh: Migrasikan database Oracle lokal Anda ke Amazon Aurora PostgreSQL Compatible Edition.
- Replatform (angkat dan bentuk ulang) — Pindahkan aplikasi ke cloud, dan perkenalkan beberapa tingkat pengoptimalan untuk memanfaatkan kemampuan cloud. Contoh: Memigrasikan database Oracle lokal Anda ke Amazon Relational Database Service (Amazon RDS) untuk Oracle di. AWS Cloud
- Pembelian kembali (drop and shop) - Beralih ke produk yang berbeda, biasanya dengan beralih dari lisensi tradisional ke model SaaS. Contoh: Migrasikan sistem manajemen hubungan pelanggan (CRM) Anda ke Salesforce.com.
- Rehost (lift dan shift) — Pindahkan aplikasi ke cloud tanpa membuat perubahan apa pun untuk memanfaatkan kemampuan cloud. Contoh: Migrasikan database Oracle lokal Anda ke Oracle pada instans EC2 di. AWS Cloud
- Relokasi (hypervisor-level lift and shift) — Pindahkan infrastruktur ke cloud tanpa membeli perangkat keras baru, menulis ulang aplikasi, atau memodifikasi operasi yang ada. Anda memigrasikan server dari platform lokal ke layanan cloud untuk platform yang sama. Contoh: Migrasikan Microsoft Hyper-V aplikasi ke AWS.
- Pertahankan (kunjungi kembali) - Simpan aplikasi di lingkungan sumber Anda. Ini mungkin termasuk aplikasi yang memerlukan refactoring besar, dan Anda ingin menunda pekerjaan itu sampai nanti, dan aplikasi lama yang ingin Anda pertahankan, karena tidak ada pembenaran bisnis untuk memigrasikannya.

- Pensiun — Menonaktifkan atau menghapus aplikasi yang tidak lagi diperlukan di lingkungan sumber Anda.

A

ABAC

Lihat kontrol akses berbasis atribut.

layanan abstrak

Lihat layanan terkelola.

ASAM

Lihat atomisitas, konsistensi, isolasi, daya tahan.

migrasi aktif-aktif

Metode migrasi database di mana database sumber dan target tetap sinkron (dengan menggunakan alat replikasi dua arah atau operasi penulisan ganda), dan kedua database menangani transaksi dari menghubungkan aplikasi selama migrasi. Metode ini mendukung migrasi dalam batch kecil yang terkontrol alih-alih memerlukan pemotongan satu kali. Ini lebih fleksibel tetapi membutuhkan lebih banyak pekerjaan daripada migrasi aktif-pasif.

migrasi aktif-pasif

Metode migrasi database di mana database sumber dan target disimpan dalam sinkron, tetapi hanya database sumber yang menangani transaksi dari menghubungkan aplikasi sementara data direplikasi ke database target. Basis data target tidak menerima transaksi apa pun selama migrasi.

fungsi agregat

Fungsi SQL yang beroperasi pada sekelompok baris dan menghitung nilai pengembalian tunggal untuk grup. Contoh fungsi agregat meliputi `SUM` dan`MAX`.

AI

Lihat kecerdasan buatan.

AIOps

Lihat operasi kecerdasan buatan.

anonimisasi

Proses menghapus informasi pribadi secara permanen dalam kumpulan data. Anonimisasi dapat membantu melindungi privasi pribadi. Data anonim tidak lagi dianggap sebagai data pribadi.

anti-pola

Solusi yang sering digunakan untuk masalah berulang di mana solusinya kontra-produktif, tidak efektif, atau kurang efektif daripada alternatif.

kontrol aplikasi

Pendekatan keamanan yang memungkinkan penggunaan hanya aplikasi yang disetujui untuk membantu melindungi sistem dari malware.

portofolio aplikasi

Kumpulan informasi rinci tentang setiap aplikasi yang digunakan oleh organisasi, termasuk biaya untuk membangun dan memelihara aplikasi, dan nilai bisnisnya. Informasi ini adalah kunci untuk penemuan portofolio dan proses analisis dan membantu mengidentifikasi dan memprioritaskan aplikasi yang akan dimigrasi, dimodernisasi, dan dioptimalkan.

kecerdasan buatan (AI)

Bidang ilmu komputer yang didedikasikan untuk menggunakan teknologi komputasi untuk melakukan fungsi kognitif yang biasanya terkait dengan manusia, seperti belajar, memecahkan masalah, dan mengenali pola. Untuk informasi lebih lanjut, lihat Apa itu Kecerdasan Buatan?

operasi kecerdasan buatan (AIOps)

Proses menggunakan teknik pembelajaran mesin untuk memecahkan masalah operasional, mengurangi insiden operasional dan intervensi manusia, dan meningkatkan kualitas layanan. Untuk informasi selengkapnya tentang cara AIOps digunakan dalam strategi AWS migrasi, lihat panduan integrasi operasi.

enkripsi asimetris

Algoritma enkripsi yang menggunakan sepasang kunci, kunci publik untuk enkripsi dan kunci pribadi untuk dekripsi. Anda dapat berbagi kunci publik karena tidak digunakan untuk dekripsi, tetapi akses ke kunci pribadi harus sangat dibatasi.

atomisitas, konsistensi, isolasi, daya tahan (ACID)

Satu set properti perangkat lunak yang menjamin validitas data dan keandalan operasional database, bahkan dalam kasus kesalahan, kegagalan daya, atau masalah lainnya.

kontrol akses berbasis atribut (ABAC)

Praktik membuat izin berbutir halus berdasarkan atribut pengguna, seperti departemen, peran pekerjaan, dan nama tim. Untuk informasi selengkapnya, lihat ABAC untuk AWS dokumentasi AWS Identity and Access Management (IAM).

sumber data otoritatif

Lokasi di mana Anda menyimpan versi utama data, yang dianggap sebagai sumber informasi yang paling dapat diandalkan. Anda dapat menyalin data dari sumber data otoritatif ke lokasi lain untuk tujuan pemrosesan atau modifikasi data, seperti menganonimkan, menyunting, atau membuat nama samaran.

Zona Ketersediaan

Lokasi berbeda di dalam Wilayah AWS yang terisolasi dari kegagalan di Availability Zone lainnya dan menyediakan konektivitas jaringan latensi rendah yang murah ke Availability Zone lainnya di Wilayah yang sama.

AWS Kerangka Adopsi Cloud (AWS CAF)

Kerangka pedoman dan praktik terbaik AWS untuk membantu organisasi mengembangkan rencana yang efisien dan efektif untuk bergerak dengan sukses ke cloud. AWS CAF mengatur panduan ke dalam enam area fokus yang disebut perspektif: bisnis, orang, tata kelola, platform, keamanan, dan operasi. Perspektif bisnis, orang, dan tata kelola fokus pada keterampilan dan proses bisnis; perspektif platform, keamanan, dan operasi fokus pada keterampilan dan proses teknis. Misalnya, perspektif masyarakat menargetkan pemangku kepentingan yang menangani sumber daya manusia (SDM), fungsi kepegawaian, dan manajemen orang. Untuk perspektif ini, AWS CAF memberikan panduan untuk pengembangan, pelatihan, dan komunikasi orang untuk membantu mempersiapkan organisasi untuk adopsi cloud yang sukses. Untuk informasi lebih lanjut, lihat situs web AWS CAF dan whitepaper AWS CAF.

AWS Kerangka Kualifikasi Beban Kerja (AWS WQF)

Alat yang mengevaluasi beban kerja migrasi database, merekomendasikan strategi migrasi, dan memberikan perkiraan kerja. AWS WQF disertakan dengan AWS Schema Conversion Tool ()AWS SCT. Ini menganalisis skema database dan objek kode, kode aplikasi, dependensi, dan karakteristik kinerja, dan memberikan laporan penilaian.

B

bot buruk

Bot yang dimaksudkan untuk mengganggu atau menyebabkan kerugian bagi individu atau organisasi.

BCP

Lihat perencanaan kontinuitas bisnis.

grafik perilaku

Pandangan interaktif yang terpadu tentang perilaku dan interaksi sumber daya dari waktu ke waktu. Anda dapat menggunakan grafik perilaku dengan Amazon Detective untuk memeriksa upaya logon yang gagal, panggilan API yang mencurigakan, dan tindakan serupa. Untuk informasi selengkapnya, lihat Data dalam grafik perilaku di dokumentasi Detektif.

sistem big-endian

Sistem yang menyimpan byte paling signifikan terlebih dahulu. Lihat juga endianness.

klasifikasi biner

Sebuah proses yang memprediksi hasil biner (salah satu dari dua kelas yang mungkin). Misalnya, model ML Anda mungkin perlu memprediksi masalah seperti “Apakah email ini spam atau bukan spam?” atau “Apakah produk ini buku atau mobil?”

filter mekar

Struktur data probabilistik dan efisien memori yang digunakan untuk menguji apakah suatu elemen adalah anggota dari suatu himpunan.

deployment biru/hijau

Strategi penyebaran tempat Anda membuat dua lingkungan yang terpisah namun identik. Anda menjalankan versi aplikasi saat ini di satu lingkungan (biru) dan versi aplikasi baru di lingkungan lain (hijau). Strategi ini membantu Anda dengan cepat memutar kembali dengan dampak minimal.

bot

Aplikasi perangkat lunak yang menjalankan tugas otomatis melalui internet dan mensimulasikan aktivitas atau interaksi manusia. Beberapa bot berguna atau bermanfaat, seperti perayap web yang mengindeks informasi di internet. Beberapa bot lain, yang dikenal sebagai bot buruk, dimaksudkan untuk mengganggu atau membahayakan individu atau organisasi.

botnet

Jaringan bot yang terinfeksi oleh malware dan berada di bawah kendali satu pihak, yang dikenal sebagai bot herder atau operator bot. Botnet adalah mekanisme paling terkenal untuk skala bot dan dampaknya.

cabang

Area berisi repositori kode. Cabang pertama yang dibuat dalam repositori adalah cabang utama. Anda dapat membuat cabang baru dari cabang yang ada, dan Anda kemudian dapat mengembangkan fitur atau memperbaiki bug di cabang baru. Cabang yang Anda buat untuk membangun fitur biasanya disebut sebagai cabang fitur. Saat fitur siap dirilis, Anda menggabungkan cabang fitur kembali ke cabang utama. Untuk informasi selengkapnya, lihat Tentang cabang (GitHub dokumentasi).

akses break-glass

Dalam keadaan luar biasa dan melalui proses yang disetujui, cara cepat bagi pengguna untuk mendapatkan akses ke Akun AWS yang biasanya tidak memiliki izin untuk mengaksesnya. Untuk informasi lebih lanjut, lihat indikator Implementasikan prosedur break-glass dalam panduan Well-Architected AWS .

strategi brownfield

Infrastruktur yang ada di lingkungan Anda. Saat mengadopsi strategi brownfield untuk arsitektur sistem, Anda merancang arsitektur di sekitar kendala sistem dan infrastruktur saat ini. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

cache penyangga

Area memori tempat data yang paling sering diakses disimpan.

kemampuan bisnis

Apa yang dilakukan bisnis untuk menghasilkan nilai (misalnya, penjualan, layanan pelanggan, atau pemasaran). Arsitektur layanan mikro dan keputusan pengembangan dapat didorong oleh kemampuan bisnis. Untuk informasi selengkapnya, lihat bagian Terorganisir di sekitar kemampuan bisnis dari Menjalankan layanan mikro kontainer di whitepaper. AWS

perencanaan kelangsungan bisnis (BCP)

Rencana yang membahas dampak potensial dari peristiwa yang mengganggu, seperti migrasi skala besar, pada operasi dan memungkinkan bisnis untuk melanjutkan operasi dengan cepat.

C

KAFE

Lihat Kerangka Adopsi AWS Cloud.

penyebaran kenari

Rilis versi yang lambat dan bertahap untuk pengguna akhir. Ketika Anda yakin, Anda menyebarkan versi baru dan mengganti versi saat ini secara keseluruhan.

CCoE

Lihat Cloud Center of Excellence.

CDC

Lihat mengubah pengambilan data.

ubah pengambilan data (CDC)

Proses melacak perubahan ke sumber data, seperti tabel database, dan merekam metadata tentang perubahan tersebut. Anda dapat menggunakan CDC untuk berbagai tujuan, seperti mengaudit atau mereplikasi perubahan dalam sistem target untuk mempertahankan sinkronisasi.

rekayasa kekacauan

Sengaja memperkenalkan kegagalan atau peristiwa yang mengganggu untuk menguji ketahanan sistem. Anda dapat menggunakan AWS Fault Injection Service (AWS FIS) untuk melakukan eksperimen yang menekankan AWS beban kerja Anda dan mengevaluasi responsnya.

CI/CD

Lihat integrasi berkelanjutan dan pengiriman berkelanjutan.

klasifikasi

Proses kategorisasi yang membantu menghasilkan prediksi. Model ML untuk masalah klasifikasi memprediksi nilai diskrit. Nilai diskrit selalu berbeda satu sama lain. Misalnya, model mungkin perlu mengevaluasi apakah ada mobil dalam gambar atau tidak.

Enkripsi sisi klien

Enkripsi data secara lokal, sebelum target Layanan AWS menerimanya.

Pusat Keunggulan Cloud (CCoE)

Tim multi-disiplin yang mendorong upaya adopsi cloud di seluruh organisasi, termasuk mengembangkan praktik terbaik cloud, memobilisasi sumber daya, menetapkan jadwal migrasi, dan memimpin organisasi melalui transformasi skala besar. Untuk informasi selengkapnya, lihat posting CCo E di Blog Strategi AWS Cloud Perusahaan.

komputasi cloud

Teknologi cloud yang biasanya digunakan untuk penyimpanan data jarak jauh dan manajemen perangkat IoT. Cloud computing umumnya terhubung ke teknologi edge computing.

model operasi cloud

Dalam organisasi TI, model operasi yang digunakan untuk membangun, mematangkan, dan mengoptimalkan satu atau lebih lingkungan cloud. Untuk informasi selengkapnya, lihat Membangun Model Operasi Cloud Anda.

tahap adopsi cloud

Empat fase yang biasanya dilalui organisasi ketika mereka bermigrasi ke AWS Cloud:

- Proyek — Menjalankan beberapa proyek terkait cloud untuk bukti konsep dan tujuan pembelajaran
- Foundation — Melakukan investasi dasar untuk meningkatkan adopsi cloud Anda (misalnya, membuat landing zone, mendefinisikan CCo E, membuat model operasi)
- Migrasi — Migrasi aplikasi individual
- Re-invention — Mengoptimalkan produk dan layanan, dan berinovasi di cloud

Tahapan ini didefinisikan oleh Stephen Orban dalam posting blog The Journey Toward Cloud-First & the Stages of Adoption di blog Strategi Perusahaan. AWS Cloud Untuk informasi tentang bagaimana kaitannya dengan strategi AWS migrasi, lihat panduan kesiapan migrasi.

CMDB

Lihat database manajemen konfigurasi.

repositori kode

Lokasi di mana kode sumber dan aset lainnya, seperti dokumentasi, sampel, dan skrip, disimpan dan diperbarui melalui proses kontrol versi. Repositori cloud umum termasuk GitHub atau. Bitbucket Cloud Setiap versi kode disebut cabang. Dalam struktur layanan mikro, setiap repositori

dikhususkan untuk satu bagian fungsionalitas. Pipa CI/CD tunggal dapat menggunakan beberapa repositori.

cache dingin

Cache buffer yang kosong, tidak terisi dengan baik, atau berisi data basi atau tidak relevan. Ini mempengaruhi kinerja karena instance database harus membaca dari memori utama atau disk, yang lebih lambat daripada membaca dari cache buffer.

data dingin

Data yang jarang diakses dan biasanya historis. Saat menanyakan jenis data ini, kueri lambat biasanya dapat diterima. Memindahkan data ini ke tingkat penyimpanan atau kelas yang berkinerja lebih rendah dan lebih murah dapat mengurangi biaya.

visi komputer (CV)

Bidang AI yang menggunakan pembelajaran mesin untuk menganalisis dan mengekstrak informasi dari format visual seperti gambar dan video digital. Misalnya, Amazon SageMaker AI menyediakan algoritma pemrosesan gambar untuk CV.

konfigurasi drift

Untuk beban kerja, konfigurasi berubah dari status yang diharapkan. Ini dapat menyebabkan beban kerja menjadi tidak patuh, dan biasanya bertahap dan tidak disengaja.

database manajemen konfigurasi (CMDB)

Repositori yang menyimpan dan mengelola informasi tentang database dan lingkungan TI, termasuk komponen perangkat keras dan perangkat lunak dan konfigurasinya. Anda biasanya menggunakan data dari CMDB dalam penemuan portofolio dan tahap analisis migrasi.

paket kesesuaian

Kumpulan AWS Config aturan dan tindakan remediasi yang dapat Anda kumpulkan untuk menyesuaikan kepatuhan dan pemeriksaan keamanan Anda. Anda dapat menerapkan paket kesesuaian sebagai entitas tunggal di Akun AWS dan Wilayah, atau di seluruh organisasi, dengan menggunakan templat YAMM. Untuk informasi selengkapnya, lihat Paket kesesuaian dalam dokumentasi. AWS Config

integrasi berkelanjutan dan pengiriman berkelanjutan (CI/CD)

Proses mengotomatiskan sumber, membangun, menguji, pementasan, dan tahap produksi dari proses rilis perangkat lunak. CI/CD biasanya digambarkan sebagai pipa. CI/CD dapat membantu

Anda mengotomatiskan proses, meningkatkan produktivitas, meningkatkan kualitas kode, dan memberikan lebih cepat. Untuk informasi lebih lanjut, lihat Manfaat pengiriman berkelanjutan. CD juga dapat berarti penerapan berkelanjutan. Untuk informasi selengkapnya, lihat Continuous Delivery vs Continuous Deployment.

CV

Lihat visi komputer.

D

data saat istirahat

Data yang stasioner di jaringan Anda, seperti data yang ada di penyimpanan.

klasifikasi data

Proses untuk mengidentifikasi dan mengkategorikan data dalam jaringan Anda berdasarkan kekritisan dan sensitivitasnya. Ini adalah komponen penting dari setiap strategi manajemen risiko keamanan siber karena membantu Anda menentukan perlindungan dan kontrol retensi yang tepat untuk data. Klasifikasi data adalah komponen pilar keamanan dalam AWS Well-Architected Framework. Untuk informasi selengkapnya, lihat Klasifikasi data.

penyimpangan data

Variasi yang berarti antara data produksi dan data yang digunakan untuk melatih model ML, atau perubahan yang berarti dalam data input dari waktu ke waktu. Penyimpangan data dapat mengurangi kualitas, akurasi, dan keadilan keseluruhan dalam prediksi model ML.

data dalam transit

Data yang aktif bergerak melalui jaringan Anda, seperti antara sumber daya jaringan.

jala data

Kerangka arsitektur yang menyediakan kepemilikan data terdistribusi dan terdesentralisasi dengan manajemen dan tata kelola terpusat.

minimalisasi data

Prinsip pengumpulan dan pemrosesan hanya data yang sangat diperlukan. Mempraktikkan minimalisasi data di dalamnya AWS Cloud dapat mengurangi risiko privasi, biaya, dan jejak karbon analitik Anda.

perimeter data

Satu set pagar pembatas pencegahan di AWS lingkungan Anda yang membantu memastikan bahwa hanya identitas tepercaya yang mengakses sumber daya tepercaya dari jaringan yang diharapkan. Untuk informasi selengkapnya, lihat Membangun perimeter data pada AWS.

prapemrosesan data

Untuk mengubah data mentah menjadi format yang mudah diuraikan oleh model ML Anda. Preprocessing data dapat berarti menghapus kolom atau baris tertentu dan menangani nilai yang hilang, tidak konsisten, atau duplikat.

asal data

Proses melacak asal dan riwayat data sepanjang siklus hidupnya, seperti bagaimana data dihasilkan, ditransmisikan, dan disimpan.

subjek data

Individu yang datanya dikumpulkan dan diproses.

gudang data

Sistem manajemen data yang mendukung intelijen bisnis, seperti analitik. Gudang data biasanya berisi sejumlah besar data historis, dan biasanya digunakan untuk kueri dan analisis.

bahasa definisi database (DDL)

Pernyataan atau perintah untuk membuat atau memodifikasi struktur tabel dan objek dalam database.

bahasa manipulasi basis data (DHTML)

Pernyataan atau perintah untuk memodifikasi (memasukkan, memperbarui, dan menghapus) informasi dalam database.

DDL

Lihat bahasa definisi database.

ansambel yang dalam

Untuk menggabungkan beberapa model pembelajaran mendalam untuk prediksi. Anda dapat menggunakan ansambel dalam untuk mendapatkan prediksi yang lebih akurat atau untuk memperkirakan ketidakpastian dalam prediksi.

pembelajaran mendalam

Subbidang ML yang menggunakan beberapa lapisan jaringan saraf tiruan untuk mengidentifikasi pemetaan antara data input dan variabel target yang diinginkan.

defense-in-depth

Pendekatan keamanan informasi di mana serangkaian mekanisme dan kontrol keamanan dilapisi dengan cermat di seluruh jaringan komputer untuk melindungi kerahasiaan, integritas, dan ketersediaan jaringan dan data di dalamnya. Saat Anda mengadopsi strategi ini AWS, Anda menambahkan beberapa kontrol pada lapisan AWS Organizations struktur yang berbeda untuk membantu mengamankan sumber daya. Misalnya, defense-in-depth pendekatan mungkin menggabungkan otentikasi multi-faktor, segmentasi jaringan, dan enkripsi.

administrator yang didelegasikan

Di AWS Organizations, layanan yang kompatibel dapat mendaftarkan akun AWS anggota untuk mengelola akun organisasi dan mengelola izin untuk layanan tersebut. Akun ini disebut administrator yang didelegasikan untuk layanan itu. Untuk informasi selengkapnya dan daftar layanan yang kompatibel, lihat Layanan yang berfungsi dengan AWS Organizations AWS Organizations dokumentasi.

deployment

Proses pembuatan aplikasi, fitur baru, atau perbaikan kode tersedia di lingkungan target. Deployment melibatkan penerapan perubahan dalam basis kode dan kemudian membangun dan menjalankan basis kode itu di lingkungan aplikasi.

lingkungan pengembangan

Lihat lingkungan.

kontrol detektif

Kontrol keamanan yang dirancang untuk mendeteksi, mencatat, dan memperingatkan setelah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan kedua, memperingatkan Anda tentang peristiwa keamanan yang melewati kontrol pencegahan yang ada. Untuk informasi selengkapnya, lihat Kontrol Detektif dalam Menerapkan kontrol keamanan pada. AWS

pemetaan aliran nilai pengembangan (DVSM)

Sebuah proses yang digunakan untuk mengidentifikasi dan memprioritaskan kendala yang mempengaruhi kecepatan dan kualitas dalam siklus hidup pengembangan perangkat lunak. DVSM memperluas proses pemetaan aliran nilai yang awalnya dirancang untuk praktik

manufaktur ramping. Ini berfokus pada langkah-langkah dan tim yang diperlukan untuk menciptakan dan memindahkan nilai melalui proses pengembangan perangkat lunak.

kembar digital

Representasi virtual dari sistem dunia nyata, seperti bangunan, pabrik, peralatan industri, atau jalur produksi. Kembar digital mendukung pemeliharaan prediktif, pemantauan jarak jauh, dan optimalisasi produksi.

tabel dimensi

Dalam skema bintang, tabel yang lebih kecil yang berisi atribut data tentang data kuantitatif dalam tabel fakta. Atribut tabel dimensi biasanya bidang teks atau angka diskrit yang berperilaku seperti teks. Atribut ini biasanya digunakan untuk pembatasan kueri, pemfilteran, dan pelabelan set hasil.

musibah

Peristiwa yang mencegah beban kerja atau sistem memenuhi tujuan bisnisnya di lokasi utama yang digunakan. Peristiwa ini dapat berupa bencana alam, kegagalan teknis, atau akibat dari tindakan manusia, seperti kesalahan konfigurasi yang tidak disengaja atau serangan malware.

pemulihan bencana (DR)

Strategi dan proses yang Anda gunakan untuk meminimalkan downtime dan kehilangan data yang disebabkan oleh bencana. Untuk informasi selengkapnya, lihat Disaster Recovery of Workloads on AWS: Recovery in the Cloud in the AWS Well-Architected Framework.

DML~

Lihat bahasa manipulasi basis data.

desain berbasis domain

Pendekatan untuk mengembangkan sistem perangkat lunak yang kompleks dengan menghubungkan komponennya ke domain yang berkembang, atau tujuan bisnis inti, yang dilayani oleh setiap komponen. Konsep ini diperkenalkan oleh Eric Evans dalam bukunya, Domain-Driven Design: Tackling Complexity in the Heart of Software (Boston: Addison-Wesley Professional, 2003). Untuk informasi tentang cara menggunakan desain berbasis domain dengan pola gambar pencekik, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

DR

Lihat pemulihan bencana.

deteksi drift

Melacak penyimpangan dari konfigurasi dasar. Misalnya, Anda dapat menggunakan AWS CloudFormation untuk mendeteksi penyimpangan dalam sumber daya sistem, atau Anda dapat menggunakannya AWS Control Tower untuk mendeteksi perubahan di landing zone yang mungkin memengaruhi kepatuhan terhadap persyaratan tata kelola.

DVSM

Lihat pemetaan aliran nilai pengembangan.

E

EDA

Lihat analisis data eksplorasi.

EDI

Lihat pertukaran data elektronik.

komputasi tepi

Teknologi yang meningkatkan daya komputasi untuk perangkat pintar di tepi jaringan IoT. Jika dibandingkan dengan komputasi awan, komputasi tepi dapat mengurangi latensi komunikasi dan meningkatkan waktu respons.

pertukaran data elektronik (EDI)

Pertukaran otomatis dokumen bisnis antar organisasi. Untuk informasi selengkapnya, lihat Apa itu Pertukaran Data Elektronik.

enkripsi

Proses komputasi yang mengubah data plaintext, yang dapat dibaca manusia, menjadi ciphertext.

kunci enkripsi

String kriptografi dari bit acak yang dihasilkan oleh algoritma enkripsi. Panjang kunci dapat bervariasi, dan setiap kunci dirancang agar tidak dapat diprediksi dan unik.

endianness

Urutan byte disimpan dalam memori komputer. Sistem big-endian menyimpan byte paling signifikan terlebih dahulu. Sistem little-endian menyimpan byte paling tidak signifikan terlebih dahulu.

titik akhir

Lihat titik akhir layanan.

layanan endpoint

Layanan yang dapat Anda host di cloud pribadi virtual (VPC) untuk dibagikan dengan pengguna lain. Anda dapat membuat layanan endpoint dengan AWS PrivateLink dan memberikan izin kepada prinsipal lain Akun AWS atau ke AWS Identity and Access Management (IAM). Akun atau prinsipal ini dapat terhubung ke layanan endpoint Anda secara pribadi dengan membuat titik akhir VPC antarmuka. Untuk informasi selengkapnya, lihat Membuat layanan titik akhir di dokumentasi Amazon Virtual Private Cloud (Amazon VPC).

perencanaan sumber daya perusahaan (ERP)

Sistem yang mengotomatiskan dan mengelola proses bisnis utama (seperti akuntansi, MES, dan manajemen proyek) untuk suatu perusahaan.

enkripsi amplop

Proses mengenkripsi kunci enkripsi dengan kunci enkripsi lain. Untuk informasi selengkapnya, lihat Enkripsi amplop dalam dokumentasi AWS Key Management Service (AWS KMS).

lingkungan

Sebuah contoh dari aplikasi yang sedang berjalan. Berikut ini adalah jenis lingkungan yang umum dalam komputasi awan:

- Development Environment — Sebuah contoh dari aplikasi yang berjalan yang hanya tersedia untuk tim inti yang bertanggung jawab untuk memelihara aplikasi. Lingkungan pengembangan digunakan untuk menguji perubahan sebelum mempromosikannya ke lingkungan atas. Jenis lingkungan ini kadang-kadang disebut sebagai lingkungan pengujian.
- lingkungan yang lebih rendah — Semua lingkungan pengembangan untuk aplikasi, seperti yang digunakan untuk build awal dan pengujian.
- lingkungan produksi — Sebuah contoh dari aplikasi yang berjalan yang pengguna akhir dapat mengakses. Dalam sebuah CI/CD pipeline, lingkungan produksi adalah lingkungan penyebaran terakhir.
- lingkungan atas — Semua lingkungan yang dapat diakses oleh pengguna selain tim pengembangan inti. Ini dapat mencakup lingkungan produksi, lingkungan praproduksi, dan lingkungan untuk pengujian penerimaan pengguna.

epik

Dalam metodologi tangkas, kategori fungsional yang membantu mengatur dan memprioritaskan pekerjaan Anda. Epik memberikan deskripsi tingkat tinggi tentang persyaratan dan tugas implementasi. Misalnya, epos keamanan AWS CAF mencakup manajemen identitas dan akses, kontrol detektif, keamanan infrastruktur, perlindungan data, dan respons insiden. Untuk informasi selengkapnya tentang epos dalam strategi AWS migrasi, lihat panduan implementasi program.

ERP

Lihat perencanaan sumber daya perusahaan.

analisis data eksplorasi (EDA)

Proses menganalisis dataset untuk memahami karakteristik utamanya. Anda mengumpulkan atau mengumpulkan data dan kemudian melakukan penyelidikan awal untuk menemukan pola, mendeteksi anomali, dan memeriksa asumsi. EDA dilakukan dengan menghitung statistik ringkasan dan membuat visualisasi data.

F

tabel fakta

Tabel tengah dalam skema bintang. Ini menyimpan data kuantitatif tentang operasi bisnis. Biasanya, tabel fakta berisi dua jenis kolom: kolom yang berisi ukuran dan yang berisi kunci asing ke tabel dimensi.

gagal cepat

Filosofi yang menggunakan pengujian yang sering dan bertahap untuk mengurangi siklus hidup pengembangan. Ini adalah bagian penting dari pendekatan tangkas.

batas isolasi kesalahan

Dalam AWS Cloud, batas seperti Availability Zone, Wilayah AWS, control plane, atau data plane yang membatasi efek kegagalan dan membantu meningkatkan ketahanan beban kerja. Untuk informasi selengkapnya, lihat Batas Isolasi AWS Kesalahan.

cabang fitur

Lihat cabang.

fitur

Data input yang Anda gunakan untuk membuat prediksi. Misalnya, dalam konteks manufaktur, fitur bisa berupa gambar yang diambil secara berkala dari lini manufaktur.

pentingnya fitur

Seberapa signifikan fitur untuk prediksi model. Ini biasanya dinyatakan sebagai skor numerik yang dapat dihitung melalui berbagai teknik, seperti Shapley Additive Explanations (SHAP) dan gradien terintegrasi. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan. AWS

transformasi fitur

Untuk mengoptimalkan data untuk proses ML, termasuk memperkaya data dengan sumber tambahan, menskalakan nilai, atau mengekstrak beberapa set informasi dari satu bidang data. Hal ini memungkinkan model ML untuk mendapatkan keuntungan dari data. Misalnya, jika Anda memecah tanggal “2021-05-27 00:15:37" menjadi “2021", “Mei”, “Kamis”, dan “15", Anda dapat membantu algoritme pembelajaran mempelajari pola bernuansa yang terkait dengan komponen data yang berbeda.

beberapa tembakan mendorong

Menyediakan LLM dengan sejumlah kecil contoh yang menunjukkan tugas dan output yang diinginkan sebelum memintanya untuk melakukan tugas serupa. Teknik ini adalah aplikasi pembelajaran dalam konteks, di mana model belajar dari contoh (bidikan) yang tertanam dalam petunjuk. Beberapa bidikan dapat efektif untuk tugas-tugas yang memerlukan pemformatan, penalaran, atau pengetahuan domain tertentu. Lihat juga zero-shot prompting.

FGAC

Lihat kontrol akses berbutir halus.

kontrol akses berbutir halus (FGAC)

Penggunaan beberapa kondisi untuk mengizinkan atau menolak permintaan akses.

migrasi flash-cut

Metode migrasi database yang menggunakan replikasi data berkelanjutan melalui pengambilan data perubahan untuk memigrasikan data dalam waktu sesingkat mungkin, alih-alih menggunakan pendekatan bertahap. Tujuannya adalah untuk menjaga downtime seminimal mungkin.

FM

Lihat model pondasi.

model pondasi (FM)

Jaringan saraf pembelajaran mendalam yang besar yang telah melatih kumpulan data besar-besaran data umum dan tidak berlabel. FMs mampu melakukan berbagai tugas umum, seperti memahami bahasa, menghasilkan teks dan gambar, dan berbicara dalam bahasa alami. Untuk informasi selengkapnya, lihat Apa itu Model Foundation.

G

AI generatif

Subset model AI yang telah dilatih pada sejumlah besar data dan yang dapat menggunakan prompt teks sederhana untuk membuat konten dan artefak baru, seperti gambar, video, teks, dan audio. Untuk informasi lebih lanjut, lihat Apa itu AI Generatif.

pemblokiran geografis

Lihat pembatasan geografis.

pembatasan geografis (pemblokiran geografis)

Di Amazon CloudFront, opsi untuk mencegah pengguna di negara tertentu mengakses distribusi konten. Anda dapat menggunakan daftar izinkan atau daftar blokir untuk menentukan negara yang disetujui dan dilarang. Untuk informasi selengkapnya, lihat Membatasi distribusi geografis konten Anda dalam dokumentasi. CloudFront

Alur kerja Gitflow

Pendekatan di mana lingkungan bawah dan atas menggunakan cabang yang berbeda dalam repositori kode sumber. Alur kerja Gitflow dianggap warisan, dan alur kerja berbasis batang adalah pendekatan modern yang lebih disukai.

gambar emas

Sebuah snapshot dari sistem atau perangkat lunak yang digunakan sebagai template untuk menyebarkan instance baru dari sistem atau perangkat lunak itu. Misalnya, di bidang manufaktur, gambar emas dapat digunakan untuk menyediakan perangkat lunak pada beberapa perangkat dan membantu meningkatkan kecepatan, skalabilitas, dan produktivitas dalam operasi manufaktur perangkat.

strategi greenfield

Tidak adanya infrastruktur yang ada di lingkungan baru. Saat mengadopsi strategi greenfield untuk arsitektur sistem, Anda dapat memilih semua teknologi baru tanpa batasan kompatibilitas dengan infrastruktur yang ada, juga dikenal sebagai brownfield. Jika Anda memperluas infrastruktur yang ada, Anda dapat memadukan strategi brownfield dan greenfield.

pagar pembatas

Aturan tingkat tinggi yang membantu mengatur sumber daya, kebijakan, dan kepatuhan di seluruh unit organisasi ()OUs. Pagar pembatas preventif menegakkan kebijakan untuk memastikan keselarasan dengan standar kepatuhan. Mereka diimplementasikan dengan menggunakan kebijakan kontrol layanan dan batas izin IAM. Detective guardrails mendeteksi pelanggaran kebijakan dan masalah kepatuhan, dan menghasilkan peringatan untuk remediasi. Mereka diimplementasikan dengan menggunakan AWS Config, AWS Security Hub CSPM, Amazon GuardDuty AWS Trusted Advisor, Amazon Inspector, dan pemeriksaan khusus AWS Lambda .

H

HA

Lihat ketersediaan tinggi.

migrasi database heterogen

Memigrasi database sumber Anda ke database target yang menggunakan mesin database yang berbeda (misalnya, Oracle ke Amazon Aurora). Migrasi heterogen biasanya merupakan bagian dari upaya arsitektur ulang, dan mengubah skema dapat menjadi tugas yang kompleks. AWS menyediakan AWS SCT yang membantu dengan konversi skema.

ketersediaan tinggi (HA)

Kemampuan beban kerja untuk beroperasi terus menerus, tanpa intervensi, jika terjadi tantangan atau bencana. Sistem HA dirancang untuk gagal secara otomatis, secara konsisten memberikan kinerja berkualitas tinggi, dan menangani beban dan kegagalan yang berbeda dengan dampak kinerja minimal.

modernisasi sejarawan

Pendekatan yang digunakan untuk memodernisasi dan meningkatkan sistem teknologi operasional (OT) untuk melayani kebutuhan industri manufaktur dengan lebih baik. Sejarawan

adalah jenis database yang digunakan untuk mengumpulkan dan menyimpan data dari berbagai sumber di pabrik.

data penahanan

Sebagian dari data historis berlabel yang ditahan dari kumpulan data yang digunakan untuk melatih model pembelajaran mesin. Anda dapat menggunakan data penahanan untuk mengevaluasi kinerja model dengan membandingkan prediksi model dengan data penahanan.

migrasi database homogen

Memigrasi database sumber Anda ke database target yang berbagi mesin database yang sama (misalnya, Microsoft SQL Server ke Amazon RDS for SQL Server). Migrasi homogen biasanya merupakan bagian dari upaya rehosting atau replatforming. Anda dapat menggunakan utilitas database asli untuk memigrasi skema.

data panas

Data yang sering diakses, seperti data real-time atau data translasi terbaru. Data ini biasanya memerlukan tingkat atau kelas penyimpanan berkinerja tinggi untuk memberikan respons kueri yang cepat.

perbaikan terbaru

Perbaikan mendesak untuk masalah kritis dalam lingkungan produksi. Karena urgensinya, perbaikan terbaru biasanya dibuat di luar alur kerja DevOps rilis biasa.

periode hypercare

Segera setelah cutover, periode waktu ketika tim migrasi mengelola dan memantau aplikasi yang dimigrasi di cloud untuk mengatasi masalah apa pun. Biasanya, periode ini panjangnya 1-4 hari. Pada akhir periode hypercare, tim migrasi biasanya mentransfer tanggung jawab untuk aplikasi ke tim operasi cloud.

I

IAc

Lihat infrastruktur sebagai kode.

kebijakan berbasis identitas

Kebijakan yang dilampirkan pada satu atau beberapa prinsip IAM yang mendefinisikan izin mereka dalam lingkungan. AWS Cloud

aplikasi idle

Aplikasi yang memiliki penggunaan CPU dan memori rata-rata antara 5 dan 20 persen selama periode 90 hari. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini atau mempertahankannya di tempat.

IIoT

Lihat Internet of Things industri.

infrastruktur yang tidak dapat diubah

Model yang menyebarkan infrastruktur baru untuk beban kerja produksi alih-alih memperbarui, menambal, atau memodifikasi infrastruktur yang ada. Infrastruktur yang tidak dapat diubah secara inheren lebih konsisten, andal, dan dapat diprediksi daripada infrastruktur yang dapat berubah. Untuk informasi selengkapnya, lihat praktik terbaik Deploy using immutable infrastructure di AWS Well-Architected Framework.

masuk (masuknya) VPC

Dalam arsitektur AWS multi-akun, VPC yang menerima, memeriksa, dan merutekan koneksi jaringan dari luar aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan inbound, outbound, dan inspeksi VPCs untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

migrasi inkremental

Strategi cutover di mana Anda memigrasikan aplikasi Anda dalam bagian-bagian kecil alih-alih melakukan satu cutover penuh. Misalnya, Anda mungkin hanya memindahkan beberapa layanan mikro atau pengguna ke sistem baru pada awalnya. Setelah Anda memverifikasi bahwa semuanya berfungsi dengan baik, Anda dapat secara bertahap memindahkan layanan mikro atau pengguna tambahan hingga Anda dapat menonaktifkan sistem lama Anda. Strategi ini mengurangi risiko yang terkait dengan migrasi besar.

Industri 4.0

Sebuah istilah yang diperkenalkan oleh Klaus Schwab pada tahun 2016 untuk merujuk pada modernisasi proses manufaktur melalui kemajuan dalam konektivitas, data real-time, otomatisasi, analitik, dan AI/ML.

infrastruktur

Semua sumber daya dan aset yang terkandung dalam lingkungan aplikasi.

infrastruktur sebagai kode (IAc)

Proses penyediaan dan pengelolaan infrastruktur aplikasi melalui satu set file konfigurasi. IAc dirancang untuk membantu Anda memusatkan manajemen infrastruktur, menstandarisasi sumber daya, dan menskalakan dengan cepat sehingga lingkungan baru dapat diulang, andal, dan konsisten.

Internet of Things industri (IIoT)

Penggunaan sensor dan perangkat yang terhubung ke internet di sektor industri, seperti manufaktur, energi, otomotif, perawatan kesehatan, ilmu kehidupan, dan pertanian. Untuk informasi lebih lanjut, lihat Membangun strategi transformasi digital Internet of Things (IIoT) industri.

inspeksi VPC

Dalam arsitektur AWS multi-akun, VPC terpusat yang mengelola inspeksi lalu lintas jaringan antara VPCs (dalam yang sama atau berbeda Wilayah AWS), internet, dan jaringan lokal. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan inbound, outbound, dan inspeksi VPCs untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

Internet of Things (IoT)

Jaringan objek fisik yang terhubung dengan sensor atau prosesor tertanam yang berkomunikasi dengan perangkat dan sistem lain melalui internet atau melalui jaringan komunikasi lokal. Untuk informasi selengkapnya, lihat Apa itu IoT?

interpretasi

Karakteristik model pembelajaran mesin yang menggambarkan sejauh mana manusia dapat memahami bagaimana prediksi model bergantung pada inputnya. Untuk informasi lebih lanjut, lihat Interpretabilitas model pembelajaran mesin dengan. AWS

IoT

Lihat Internet of Things.

Perpustakaan informasi TI (ITIL)

Serangkaian praktik terbaik untuk memberikan layanan TI dan menyelaraskan layanan ini dengan persyaratan bisnis. ITIL menyediakan dasar untuk ITSM.

Manajemen layanan TI (ITSM)

Kegiatan yang terkait dengan merancang, menerapkan, mengelola, dan mendukung layanan TI untuk suatu organisasi. Untuk informasi tentang mengintegrasikan operasi cloud dengan alat ITSM, lihat panduan integrasi operasi.

ITIL

Lihat perpustakaan informasi TI.

ITSM

Lihat manajemen layanan TI.

L

kontrol akses berbasis label (LBAC)

Implementasi kontrol akses wajib (MAC) di mana pengguna dan data itu sendiri masing-masing secara eksplisit diberi nilai label keamanan. Persimpangan antara label keamanan pengguna dan label keamanan data menentukan baris dan kolom mana yang dapat dilihat oleh pengguna.

landing zone

Landing zone adalah AWS lingkungan multi-akun yang dirancang dengan baik yang dapat diskalakan dan aman. Ini adalah titik awal dari mana organisasi Anda dapat dengan cepat meluncurkan dan menyebarkan beban kerja dan aplikasi dengan percaya diri dalam lingkungan keamanan dan infrastruktur mereka. Untuk informasi selengkapnya tentang zona pendaratan, lihat Menyiapkan lingkungan multi-akun AWS yang aman dan dapat diskalakan.

model bahasa besar (LLM)

Model AI pembelajaran mendalam yang dilatih sebelumnya pada sejumlah besar data. LLM dapat melakukan beberapa tugas, seperti menjawab pertanyaan, meringkas dokumen, menerjemahkan teks ke dalam bahasa lain, dan menyelesaikan kalimat. Untuk informasi lebih lanjut, lihat Apa itu LLMs.

migrasi besar

Migrasi 300 atau lebih server.

LBAC

Lihat kontrol akses berbasis label.

hak istimewa paling sedikit

Praktik keamanan terbaik untuk memberikan izin minimum yang diperlukan untuk melakukan tugas. Untuk informasi selengkapnya, lihat Menerapkan izin hak istimewa terkecil dalam dokumentasi IAM.

angkat dan geser

Lihat 7 Rs.

sistem endian kecil

Sebuah sistem yang menyimpan byte paling tidak signifikan terlebih dahulu. Lihat juga endianness.

LLM

Lihat model bahasa besar.

lingkungan yang lebih rendah

Lihat lingkungan.

M

pembelajaran mesin (ML)

Jenis kecerdasan buatan yang menggunakan algoritma dan teknik untuk pengenalan pola dan pembelajaran. ML menganalisis dan belajar dari data yang direkam, seperti data Internet of Things (IoT), untuk menghasilkan model statistik berdasarkan pola. Untuk informasi selengkapnya, lihat Machine Learning.

cabang utama

Lihat cabang.

malware

Perangkat lunak yang dirancang untuk membahayakan keamanan atau privasi komputer. Malware dapat mengganggu sistem komputer, membocorkan informasi sensitif, atau mendapatkan akses yang tidak sah. Contoh malware termasuk virus, worm, ransomware, Trojan horse, spyware, dan keyloggers.

layanan terkelola

Layanan AWS yang AWS mengoperasikan lapisan infrastruktur, sistem operasi, dan platform, dan Anda mengakses titik akhir untuk menyimpan dan mengambil data. Amazon Simple Storage Service (Amazon S3) dan Amazon DynamoDB adalah contoh layanan terkelola. Ini juga dikenal sebagai layanan abstrak.

sistem eksekusi manufaktur (MES)

Sistem perangkat lunak untuk melacak, memantau, mendokumentasikan, dan mengendalikan proses produksi yang mengubah bahan baku menjadi produk jadi di lantai toko.

PETA

Lihat Program Percepatan Migrasi.

mekanisme

Proses lengkap di mana Anda membuat alat, mendorong adopsi alat, dan kemudian memeriksa hasilnya untuk melakukan penyesuaian. Mekanisme adalah siklus yang memperkuat dan meningkatkan dirinya sendiri saat beroperasi. Untuk informasi lebih lanjut, lihat Membangun mekanisme di AWS Well-Architected Framework.

akun anggota

Semua Akun AWS selain akun manajemen yang merupakan bagian dari organisasi di AWS Organizations. Akun dapat menjadi anggota dari hanya satu organisasi pada suatu waktu.

MES

Lihat sistem eksekusi manufaktur.

Transportasi Telemetri Antrian Pesan (MQTT)

Protokol komunikasi ringan machine-to-machine (M2M), berdasarkan pola terbitkan/berlangganan, untuk perangkat IoT yang dibatasi sumber daya.

layanan mikro

Layanan kecil dan independen yang berkomunikasi dengan jelas APIs dan biasanya dimiliki oleh tim kecil yang mandiri. Misalnya, sistem asuransi mungkin mencakup layanan mikro yang memetakan kemampuan bisnis, seperti penjualan atau pemasaran, atau subdomain, seperti pembelian, klaim, atau analitik. Manfaat layanan mikro termasuk kelincahan, penskalaan yang fleksibel, penyebaran yang mudah, kode yang dapat digunakan kembali, dan ketahanan. Untuk

informasi selengkapnya, lihat Mengintegrasikan layanan mikro dengan menggunakan layanan tanpa AWS server.

arsitektur microservices

Pendekatan untuk membangun aplikasi dengan komponen independen yang menjalankan setiap proses aplikasi sebagai layanan mikro. Layanan mikro ini berkomunikasi melalui antarmuka yang terdefinisi dengan baik dengan menggunakan ringan. APIs Setiap layanan mikro dalam arsitektur ini dapat diperbarui, digunakan, dan diskalakan untuk memenuhi permintaan fungsi tertentu dari suatu aplikasi. Untuk informasi selengkapnya, lihat Menerapkan layanan mikro di AWS.

Program Percepatan Migrasi (MAP)

AWS Program yang menyediakan dukungan konsultasi, pelatihan, dan layanan untuk membantu organisasi membangun fondasi operasional yang kuat untuk pindah ke cloud, dan untuk membantu mengimbangi biaya awal migrasi. MAP mencakup metodologi migrasi untuk mengeksekusi migrasi lama dengan cara metodis dan seperangkat alat untuk mengotomatisasi dan mempercepat skenario migrasi umum.

migrasi dalam skala

Proses memindahkan sebagian besar portofolio aplikasi ke cloud dalam gelombang, dengan lebih banyak aplikasi bergerak pada tingkat yang lebih cepat di setiap gelombang. Fase ini menggunakan praktik dan pelajaran terbaik dari fase sebelumnya untuk mengimplementasikan pabrik migrasi tim, alat, dan proses untuk merampingkan migrasi beban kerja melalui otomatisasi dan pengiriman tangkas. Ini adalah fase ketiga dari strategi AWS migrasi.

pabrik migrasi

Tim lintas fungsi yang merampingkan migrasi beban kerja melalui pendekatan otomatis dan gesit. Tim pabrik migrasi biasanya mencakup operasi, analis dan pemilik bisnis, insinyur migrasi, pengembang, dan DevOps profesional yang bekerja di sprint. Antara 20 dan 50 persen portofolio aplikasi perusahaan terdiri dari pola berulang yang dapat dioptimalkan dengan pendekatan pabrik. Untuk informasi selengkapnya, lihat diskusi tentang pabrik migrasi dan panduan Pabrik Migrasi Cloud di kumpulan konten ini.

metadata migrasi

Informasi tentang aplikasi dan server yang diperlukan untuk menyelesaikan migrasi. Setiap pola migrasi memerlukan satu set metadata migrasi yang berbeda. Contoh metadata migrasi termasuk subnet target, grup keamanan, dan akun. AWS

pola migrasi

Tugas migrasi berulang yang merinci strategi migrasi, tujuan migrasi, dan aplikasi atau layanan migrasi yang digunakan. Contoh: Rehost migrasi ke Amazon EC2 AWS dengan Layanan Migrasi Aplikasi.

Penilaian Portofolio Migrasi (MPA)

Alat online yang menyediakan informasi untuk memvalidasi kasus bisnis untuk bermigrasi ke. AWS Cloud MPA menyediakan penilaian portofolio terperinci (ukuran kanan server, harga, perbandingan TCO, analisis biaya migrasi) serta perencanaan migrasi (analisis data aplikasi dan pengumpulan data, pengelompokan aplikasi, prioritas migrasi, dan perencanaan gelombang). Alat MPA (memerlukan login) tersedia gratis untuk semua AWS konsultan dan konsultan APN Partner.

Penilaian Kesiapan Migrasi (MRA)

Proses mendapatkan wawasan tentang status kesiapan cloud organisasi, mengidentifikasi kekuatan dan kelemahan, dan membangun rencana aksi untuk menutup kesenjangan yang diidentifikasi, menggunakan CAF. AWS Untuk informasi selengkapnya, lihat panduan kesiapan migrasi. MRA adalah tahap pertama dari strategi AWS migrasi.

strategi migrasi

Pendekatan yang digunakan untuk memigrasikan beban kerja ke file. AWS Cloud Untuk informasi lebih lanjut, lihat entri 7 Rs di glosarium ini dan lihat Memobilisasi organisasi Anda untuk mempercepat migrasi skala besar.

ML

Lihat pembelajaran mesin.

modernisasi

Mengubah aplikasi usang (warisan atau monolitik) dan infrastrukturnya menjadi sistem yang gesit, elastis, dan sangat tersedia di cloud untuk mengurangi biaya, mendapatkan efisiensi, dan memanfaatkan inovasi. Untuk informasi selengkapnya, lihat Strategi untuk memodernisasi aplikasi di. AWS Cloud

penilaian kesiapan modernisasi

Evaluasi yang membantu menentukan kesiapan modernisasi aplikasi organisasi; mengidentifikasi manfaat, risiko, dan dependensi; dan menentukan seberapa baik organisasi dapat mendukung keadaan masa depan aplikasi tersebut. Hasil penilaian adalah cetak biru arsitektur target, peta

jalan yang merinci fase pengembangan dan tonggak untuk proses modernisasi, dan rencana aksi untuk mengatasi kesenjangan yang diidentifikasi. Untuk informasi lebih lanjut, lihat Mengevaluasi kesiapan modernisasi untuk aplikasi di. AWS Cloud

aplikasi monolitik (monolit)

Aplikasi yang berjalan sebagai layanan tunggal dengan proses yang digabungkan secara ketat. Aplikasi monolitik memiliki beberapa kelemahan. Jika satu fitur aplikasi mengalami lonjakan permintaan, seluruh arsitektur harus diskalakan. Menambahkan atau meningkatkan fitur aplikasi monolitik juga menjadi lebih kompleks ketika basis kode tumbuh. Untuk mengatasi masalah ini, Anda dapat menggunakan arsitektur microservices. Untuk informasi lebih lanjut, lihat Mengurai monolit menjadi layanan mikro.

MPA

Lihat Penilaian Portofolio Migrasi.

MQTT

Lihat Transportasi Telemetri Antrian Pesan.

klasifikasi multiclass

Sebuah proses yang membantu menghasilkan prediksi untuk beberapa kelas (memprediksi satu dari lebih dari dua hasil). Misalnya, model ML mungkin bertanya “Apakah produk ini buku, mobil, atau telepon?” atau “Kategori produk mana yang paling menarik bagi pelanggan ini?”

infrastruktur yang bisa berubah

Model yang memperbarui dan memodifikasi infrastruktur yang ada untuk beban kerja produksi. Untuk meningkatkan konsistensi, keandalan, dan prediktabilitas, AWS Well-Architected Framework merekomendasikan penggunaan infrastruktur yang tidak dapat diubah sebagai praktik terbaik.

O

OAC

Lihat kontrol akses asal.

OAI

Lihat identitas akses asal.

OCM

Lihat manajemen perubahan organisasi.

migrasi offline

Metode migrasi di mana beban kerja sumber diturunkan selama proses migrasi. Metode ini melibatkan waktu henti yang diperpanjang dan biasanya digunakan untuk beban kerja kecil dan tidak kritis.

OI

Lihat integrasi operasi.

OLA

Lihat perjanjian tingkat operasional.

migrasi online

Metode migrasi di mana beban kerja sumber disalin ke sistem target tanpa diambil offline. Aplikasi yang terhubung ke beban kerja dapat terus berfungsi selama migrasi. Metode ini melibatkan waktu henti nol hingga minimal dan biasanya digunakan untuk beban kerja produksi yang kritis.

OPC-UA

Lihat Komunikasi Proses Terbuka - Arsitektur Terpadu.

Komunikasi Proses Terbuka - Arsitektur Terpadu (OPC-UA)

Protokol komunikasi machine-to-machine (M2M) untuk otomasi industri. OPC-UA menyediakan standar interoperabilitas dengan enkripsi data, otentikasi, dan skema otorisasi.

perjanjian tingkat operasional (OLA)

Perjanjian yang menjelaskan apa yang dijanjikan kelompok TI fungsional untuk diberikan satu sama lain, untuk mendukung perjanjian tingkat layanan (SLA).

Tinjauan Kesiapan Operasional (ORR)

Daftar pertanyaan dan praktik terbaik terkait yang membantu Anda memahami, mengevaluasi, mencegah, atau mengurangi ruang lingkup insiden dan kemungkinan kegagalan. Untuk informasi lebih lanjut, lihat Ulasan Kesiapan Operasional (ORR) dalam Kerangka Kerja Well-Architected AWS .

teknologi operasional (OT)

Sistem perangkat keras dan perangkat lunak yang bekerja dengan lingkungan fisik untuk mengendalikan operasi industri, peralatan, dan infrastruktur. Di bidang manufaktur, integrasi sistem OT dan teknologi informasi (TI) adalah fokus utama untuk transformasi Industri 4.0.

integrasi operasi (OI)

Proses modernisasi operasi di cloud, yang melibatkan perencanaan kesiapan, otomatisasi, dan integrasi. Untuk informasi selengkapnya, lihat panduan integrasi operasi.

jejak organisasi

Jejak yang dibuat oleh AWS CloudTrail itu mencatat semua peristiwa untuk semua Akun AWS dalam organisasi di AWS Organizations. Jejak ini dibuat di setiap Akun AWS bagian organisasi dan melacak aktivitas di setiap akun. Untuk informasi selengkapnya, lihat Membuat jejak untuk organisasi dalam CloudTrail dokumentasi.

manajemen perubahan organisasi (OCM)

Kerangka kerja untuk mengelola transformasi bisnis utama yang mengganggu dari perspektif orang, budaya, dan kepemimpinan. OCM membantu organisasi mempersiapkan, dan transisi ke, sistem dan strategi baru dengan mempercepat adopsi perubahan, mengatasi masalah transisi, dan mendorong perubahan budaya dan organisasi. Dalam strategi AWS migrasi, kerangka kerja ini disebut percepatan orang, karena kecepatan perubahan yang diperlukan dalam proyek adopsi cloud. Untuk informasi lebih lanjut, lihat panduan OCM.

kontrol akses asal (OAC)

Di CloudFront, opsi yang disempurnakan untuk membatasi akses untuk mengamankan konten Amazon Simple Storage Service (Amazon S3) Anda. OAC mendukung semua bucket S3 di semua Wilayah AWS, enkripsi sisi server dengan AWS KMS (SSE-KMS), dan dinamis dan permintaan ke bucket S3. `PUT` `DELETE`

identitas akses asal (OAI)

Di CloudFront, opsi untuk membatasi akses untuk mengamankan konten Amazon S3 Anda. Saat Anda menggunakan OAI, CloudFront buat prinsipal yang dapat diautentikasi oleh Amazon S3. Prinsipal yang diautentikasi dapat mengakses konten dalam bucket S3 hanya melalui distribusi tertentu. CloudFront Lihat juga OAC, yang menyediakan kontrol akses yang lebih terperinci dan ditingkatkan.

ORR

Lihat tinjauan kesiapan operasional.

OT

Lihat teknologi operasional.

keluar (jalan keluar) VPC

Dalam arsitektur AWS multi-akun, VPC yang menangani koneksi jaringan yang dimulai dari dalam aplikasi. Arsitektur Referensi AWS Keamanan merekomendasikan pengaturan akun Jaringan Anda dengan inbound, outbound, dan inspeksi VPCs untuk melindungi antarmuka dua arah antara aplikasi Anda dan internet yang lebih luas.

P

batas izin

Kebijakan manajemen IAM yang dilampirkan pada prinsipal IAM untuk menetapkan izin maksimum yang dapat dimiliki pengguna atau peran. Untuk informasi selengkapnya, lihat Batas izin dalam dokumentasi IAM.

Informasi Identifikasi Pribadi (PII)

Informasi yang, jika dilihat secara langsung atau dipasangkan dengan data terkait lainnya, dapat digunakan untuk menyimpulkan identitas individu secara wajar. Contoh PII termasuk nama, alamat, dan informasi kontak.

PII

Lihat informasi yang dapat diidentifikasi secara pribadi.

buku pedoman

Serangkaian langkah yang telah ditentukan sebelumnya yang menangkap pekerjaan yang terkait dengan migrasi, seperti mengirimkan fungsi operasi inti di cloud. Buku pedoman dapat berupa skrip, runbook otomatis, atau ringkasan proses atau langkah-langkah yang diperlukan untuk mengoperasikan lingkungan modern Anda.

PLC

Lihat pengontrol logika yang dapat diprogram.

PLM

Lihat manajemen siklus hidup produk.

kebijakan

Objek yang dapat menentukan izin (lihat kebijakan berbasis identitas), menentukan kondisi akses (lihat kebijakan berbasis sumber daya), atau menentukan izin maksimum untuk semua akun di organisasi (lihat kebijakan kontrol layanan). AWS Organizations

persistensi poliglot

Secara independen memilih teknologi penyimpanan data microservice berdasarkan pola akses data dan persyaratan lainnya. Jika layanan mikro Anda memiliki teknologi penyimpanan data yang sama, mereka dapat menghadapi tantangan implementasi atau mengalami kinerja yang buruk. Layanan mikro lebih mudah diimplementasikan dan mencapai kinerja dan skalabilitas yang lebih baik jika mereka menggunakan penyimpanan data yang paling sesuai dengan kebutuhan mereka.

penilaian portofolio

Proses menemukan, menganalisis, dan memprioritaskan portofolio aplikasi untuk merencanakan migrasi. Untuk informasi selengkapnya, lihat Mengevaluasi kesiapan migrasi.

predikat

Kondisi kueri yang mengembalikan `true` atau`false`, biasanya terletak di `WHERE` klausa.

predikat pushdown

Teknik optimasi kueri database yang menyaring data dalam kueri sebelum transfer. Ini mengurangi jumlah data yang harus diambil dan diproses dari database relasional, dan meningkatkan kinerja kueri.

kontrol preventif

Kontrol keamanan yang dirancang untuk mencegah suatu peristiwa terjadi. Kontrol ini adalah garis pertahanan pertama untuk membantu mencegah akses tidak sah atau perubahan yang tidak diinginkan ke jaringan Anda. Untuk informasi selengkapnya, lihat Kontrol pencegahan dalam Menerapkan kontrol keamanan pada. AWS

principal

Entitas AWS yang dapat melakukan tindakan dan mengakses sumber daya. Entitas ini biasanya merupakan pengguna root untuk Akun AWS, peran IAM, atau pengguna. Untuk informasi selengkapnya, lihat Prinsip dalam istilah dan konsep Peran dalam dokumentasi IAM.

privasi berdasarkan desain

Pendekatan rekayasa sistem yang memperhitungkan privasi melalui seluruh proses pengembangan.

zona host pribadi

Container yang menyimpan informasi tentang bagaimana Anda ingin Amazon Route 53 merespons kueri DNS untuk domain dan subdomainnya dalam satu atau lebih. VPCs Untuk informasi selengkapnya, lihat Bekerja dengan zona yang dihosting pribadi di dokumentasi Route 53.

kontrol proaktif

Kontrol keamanan yang dirancang untuk mencegah penyebaran sumber daya yang tidak sesuai. Kontrol ini memindai sumber daya sebelum disediakan. Jika sumber daya tidak sesuai dengan kontrol, maka itu tidak disediakan. Untuk informasi selengkapnya, lihat panduan referensi Kontrol dalam AWS Control Tower dokumentasi dan lihat Kontrol proaktif dalam Menerapkan kontrol keamanan pada AWS.

manajemen siklus hidup produk (PLM)

Manajemen data dan proses untuk suatu produk di seluruh siklus hidupnya, mulai dari desain, pengembangan, dan peluncuran, melalui pertumbuhan dan kematangan, hingga penurunan dan penghapusan.

lingkungan produksi

Lihat lingkungan.

pengontrol logika yang dapat diprogram (PLC)

Di bidang manufaktur, komputer yang sangat andal dan mudah beradaptasi yang memantau mesin dan mengotomatiskan proses manufaktur.

rantai cepat

Menggunakan output dari satu prompt LLM sebagai input untuk prompt berikutnya untuk menghasilkan respons yang lebih baik. Teknik ini digunakan untuk memecah tugas yang kompleks menjadi subtugas, atau untuk secara iteratif memperbaiki atau memperluas respons awal. Ini membantu meningkatkan akurasi dan relevansi respons model dan memungkinkan hasil yang lebih terperinci dan dipersonalisasi.

pseudonimisasi

Proses penggantian pengidentifikasi pribadi dalam kumpulan data dengan nilai placeholder. Pseudonimisasi dapat membantu melindungi privasi pribadi. Data pseudonim masih dianggap sebagai data pribadi.

publish/subscribe (pub/sub)

Pola yang memungkinkan komunikasi asinkron antara layanan mikro untuk meningkatkan skalabilitas dan daya tanggap. Misalnya, dalam MES berbasis layanan mikro, layanan mikro dapat mempublikasikan pesan peristiwa ke saluran yang dapat berlangganan layanan mikro lainnya. Sistem dapat menambahkan layanan mikro baru tanpa mengubah layanan penerbitan.

Q

rencana kueri

Serangkaian langkah, seperti instruksi, yang digunakan untuk mengakses data dalam sistem database relasional SQL.

regresi rencana kueri

Ketika pengoptimal layanan database memilih rencana yang kurang optimal daripada sebelum perubahan yang diberikan ke lingkungan database. Hal ini dapat disebabkan oleh perubahan statistik, kendala, pengaturan lingkungan, pengikatan parameter kueri, dan pembaruan ke mesin database.

R

Matriks RACI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

LAP

Lihat Retrieval Augmented Generation.

ransomware

Perangkat lunak berbahaya yang dirancang untuk memblokir akses ke sistem komputer atau data sampai pembayaran dilakukan.

Matriks RASCI

Lihat bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI).

RCAC

Lihat kontrol akses baris dan kolom.

replika baca

Salinan database yang digunakan untuk tujuan read-only. Anda dapat merutekan kueri ke replika baca untuk mengurangi beban pada database utama Anda.

arsitek ulang

Lihat 7 Rs.

tujuan titik pemulihan (RPO)

Jumlah waktu maksimum yang dapat diterima sejak titik pemulihan data terakhir. Ini menentukan apa yang dianggap sebagai kehilangan data yang dapat diterima antara titik pemulihan terakhir dan gangguan layanan.

tujuan waktu pemulihan (RTO)

Penundaan maksimum yang dapat diterima antara gangguan layanan dan pemulihan layanan.

refactor

Lihat 7 Rs.

Region

Kumpulan AWS sumber daya di wilayah geografis. Masing-masing Wilayah AWS terisolasi dan independen dari yang lain untuk memberikan toleransi kesalahan, stabilitas, dan ketahanan. Untuk informasi selengkapnya, lihat Menentukan Wilayah AWS akun yang dapat digunakan.

regresi

Teknik ML yang memprediksi nilai numerik. Misalnya, untuk memecahkan masalah “Berapa harga rumah ini akan dijual?” Model ML dapat menggunakan model regresi linier untuk memprediksi harga jual rumah berdasarkan fakta yang diketahui tentang rumah (misalnya, luas persegi).

rehost

Lihat 7 Rs.

melepaskan

Dalam proses penyebaran, tindakan mempromosikan perubahan pada lingkungan produksi.

memindahkan

Lihat 7 Rs.

memplatform ulang

Lihat 7 Rs.

pembelian kembali

Lihat 7 Rs.

ketahanan

Kemampuan aplikasi untuk melawan atau pulih dari gangguan. Ketersediaan tinggi dan pemulihan bencana adalah pertimbangan umum ketika merencanakan ketahanan di. AWS Cloud Untuk informasi lebih lanjut, lihat AWS Cloud Ketahanan.

kebijakan berbasis sumber daya

Kebijakan yang dilampirkan ke sumber daya, seperti bucket Amazon S3, titik akhir, atau kunci enkripsi. Jenis kebijakan ini menentukan prinsipal mana yang diizinkan mengakses, tindakan yang didukung, dan kondisi lain yang harus dipenuhi.

matriks yang bertanggung jawab, akuntabel, dikonsultasikan, diinformasikan (RACI)

Matriks yang mendefinisikan peran dan tanggung jawab untuk semua pihak yang terlibat dalam kegiatan migrasi dan operasi cloud. Nama matriks berasal dari jenis tanggung jawab yang didefinisikan dalam matriks: bertanggung jawab (R), akuntabel (A), dikonsultasikan (C), dan diinformasikan (I). Jenis dukungan (S) adalah opsional. Jika Anda menyertakan dukungan, matriks disebut matriks RASCI, dan jika Anda mengecualikannya, itu disebut matriks RACI.

kontrol responsif

Kontrol keamanan yang dirancang untuk mendorong remediasi efek samping atau penyimpangan dari garis dasar keamanan Anda. Untuk informasi selengkapnya, lihat Kontrol responsif dalam Menerapkan kontrol keamanan pada AWS.

melestarikan

Lihat 7 Rs.

pensiun

Lihat 7 Rs.

Retrieval Augmented Generation (RAG)

Teknologi AI generatif di mana LLM mereferensikan sumber data otoritatif yang berada di luar sumber data pelatihannya sebelum menghasilkan respons. Misalnya, model RAG mungkin

melakukan pencarian semantik dari basis pengetahuan organisasi atau data kustom. Untuk informasi lebih lanjut, lihat Apa itu RAG.

rotasi

Proses memperbarui rahasia secara berkala untuk membuatnya lebih sulit bagi penyerang untuk mengakses kredensil.

kontrol akses baris dan kolom (RCAC)

Penggunaan ekspresi SQL dasar dan fleksibel yang telah menetapkan aturan akses. RCAC terdiri dari izin baris dan topeng kolom.

RPO

Lihat tujuan titik pemulihan.

RTO

Lihat tujuan waktu pemulihan.

buku runbook

Satu set prosedur manual atau otomatis yang diperlukan untuk melakukan tugas tertentu. Ini biasanya dibangun untuk merampingkan operasi berulang atau prosedur dengan tingkat kesalahan yang tinggi.

D

SAML 2.0

Standar terbuka yang digunakan oleh banyak penyedia identitas (IdPs). Fitur ini memungkinkan sistem masuk tunggal gabungan (SSO), sehingga pengguna dapat masuk ke Konsol Manajemen AWS atau memanggil operasi AWS API tanpa Anda harus membuat pengguna di IAM untuk semua orang di organisasi Anda. Untuk informasi lebih lanjut tentang federasi berbasis SAMP 2.0, lihat Tentang federasi berbasis SAMP 2.0 dalam dokumentasi IAM.

PENIPUAN

Lihat kontrol pengawasan dan akuisisi data.

SCP

Lihat kebijakan kontrol layanan.

Rahasia

Dalam AWS Secrets Manager, informasi rahasia atau terbatas, seperti kata sandi atau kredensi pengguna, yang Anda simpan dalam bentuk terenkripsi. Ini terdiri dari nilai rahasia dan metadata-nya. Nilai rahasia dapat berupa biner, string tunggal, atau beberapa string. Untuk informasi selengkapnya, lihat Apa yang ada di rahasia Secrets Manager? dalam dokumentasi Secrets Manager.

keamanan dengan desain

Pendekatan rekayasa sistem yang memperhitungkan keamanan melalui seluruh proses pengembangan.

kontrol keamanan

Pagar pembatas teknis atau administratif yang mencegah, mendeteksi, atau mengurangi kemampuan pelaku ancaman untuk mengeksploitasi kerentanan keamanan. Ada empat jenis kontrol keamanan utama: preventif, detektif, responsif, dan proaktif.

pengerasan keamanan

Proses mengurangi permukaan serangan untuk membuatnya lebih tahan terhadap serangan. Ini dapat mencakup tindakan seperti menghapus sumber daya yang tidak lagi diperlukan, menerapkan praktik keamanan terbaik untuk memberikan hak istimewa paling sedikit, atau menonaktifkan fitur yang tidak perlu dalam file konfigurasi.

sistem informasi keamanan dan manajemen acara (SIEM)

Alat dan layanan yang menggabungkan sistem manajemen informasi keamanan (SIM) dan manajemen acara keamanan (SEM). Sistem SIEM mengumpulkan, memantau, dan menganalisis data dari server, jaringan, perangkat, dan sumber lain untuk mendeteksi ancaman dan pelanggaran keamanan, dan untuk menghasilkan peringatan.

otomatisasi respons keamanan

Tindakan yang telah ditentukan dan diprogram yang dirancang untuk secara otomatis merespons atau memulihkan peristiwa keamanan. Otomatisasi ini berfungsi sebagai kontrol keamanan detektif atau responsif yang membantu Anda menerapkan praktik terbaik AWS keamanan. Contoh tindakan respons otomatis termasuk memodifikasi grup keamanan VPC, menambal instans Amazon EC2, atau memutar kredensil.

enkripsi sisi server

Enkripsi data di tujuannya, oleh Layanan AWS yang menerimanya.

kebijakan kontrol layanan (SCP)

Kebijakan yang menyediakan kontrol terpusat atas izin untuk semua akun di organisasi. AWS Organizations SCPs menentukan pagar pembatas atau menetapkan batasan pada tindakan yang dapat didelegasikan oleh administrator kepada pengguna atau peran. Anda dapat menggunakan SCPs daftar izin atau daftar penolakan, untuk menentukan layanan atau tindakan mana yang diizinkan atau dilarang. Untuk informasi selengkapnya, lihat Kebijakan kontrol layanan dalam AWS Organizations dokumentasi.

titik akhir layanan

URL titik masuk untuk file Layanan AWS. Anda dapat menggunakan endpoint untuk terhubung secara terprogram ke layanan target. Untuk informasi selengkapnya, lihat Layanan AWS titik akhir di Referensi Umum AWS.

perjanjian tingkat layanan (SLA)

Perjanjian yang menjelaskan apa yang dijanjikan tim TI untuk diberikan kepada pelanggan mereka, seperti waktu kerja dan kinerja layanan.

indikator tingkat layanan (SLI)

Pengukuran aspek kinerja layanan, seperti tingkat kesalahan, ketersediaan, atau throughputnya.

tujuan tingkat layanan (SLO)

Metrik target yang mewakili kesehatan layanan, yang diukur dengan indikator tingkat layanan.

model tanggung jawab bersama

Model yang menjelaskan tanggung jawab yang Anda bagikan AWS untuk keamanan dan kepatuhan cloud. AWS bertanggung jawab atas keamanan cloud, sedangkan Anda bertanggung jawab atas keamanan di cloud. Untuk informasi selengkapnya, lihat Model tanggung jawab bersama.

SIEM

Lihat informasi keamanan dan sistem manajemen acara.

titik kegagalan tunggal (SPOF)

Kegagalan dalam satu komponen penting dari aplikasi yang dapat mengganggu sistem.

SLA

Lihat perjanjian tingkat layanan.

SLI

Lihat indikator tingkat layanan.

SLO

Lihat tujuan tingkat layanan.

split-and-seed model

Pola untuk menskalakan dan mempercepat proyek modernisasi. Ketika fitur baru dan rilis produk didefinisikan, tim inti berpisah untuk membuat tim produk baru. Ini membantu meningkatkan kemampuan dan layanan organisasi Anda, meningkatkan produktivitas pengembang, dan mendukung inovasi yang cepat. Untuk informasi lebih lanjut, lihat Pendekatan bertahap untuk memodernisasi aplikasi di. AWS Cloud

SPOF

Lihat satu titik kegagalan.

skema bintang

Struktur organisasi database yang menggunakan satu tabel fakta besar untuk menyimpan data transaksional atau terukur dan menggunakan satu atau lebih tabel dimensi yang lebih kecil untuk menyimpan atribut data. Struktur ini dirancang untuk digunakan dalam gudang data atau untuk tujuan intelijen bisnis.

pola ara pencekik

Pendekatan untuk memodernisasi sistem monolitik dengan menulis ulang secara bertahap dan mengganti fungsionalitas sistem sampai sistem warisan dapat dinonaktifkan. Pola ini menggunakan analogi pohon ara yang tumbuh menjadi pohon yang sudah mapan dan akhirnya mengatasi dan menggantikan inangnya. Pola ini diperkenalkan oleh Martin Fowler sebagai cara untuk mengelola risiko saat menulis ulang sistem monolitik. Untuk contoh cara menerapkan pola ini, lihat Memodernisasi layanan web Microsoft ASP.NET (ASMX) lama secara bertahap menggunakan container dan Amazon API Gateway.

subnet

Rentang alamat IP dalam VPC Anda. Subnet harus berada di Availability Zone tunggal.

kontrol pengawasan dan akuisisi data (SCADA)

Di bidang manufaktur, sistem yang menggunakan perangkat keras dan perangkat lunak untuk memantau aset fisik dan operasi produksi.

enkripsi simetris

Algoritma enkripsi yang menggunakan kunci yang sama untuk mengenkripsi dan mendekripsi data.

pengujian sintetis

Menguji sistem dengan cara yang mensimulasikan interaksi pengguna untuk mendeteksi potensi masalah atau untuk memantau kinerja. Anda dapat menggunakan Amazon CloudWatch Synthetics untuk membuat tes ini.

sistem prompt

Teknik untuk memberikan konteks, instruksi, atau pedoman ke LLM untuk mengarahkan perilakunya. Permintaan sistem membantu mengatur konteks dan menetapkan aturan untuk interaksi dengan pengguna.

T

tag

Pasangan nilai kunci yang bertindak sebagai metadata untuk mengatur sumber daya Anda. AWS Tanda membantu Anda mengelola, mengidentifikasi, mengatur, dan memfilter sumber daya. Untuk informasi selengkapnya, lihat Menandai sumber daya AWS.

variabel target

Nilai yang Anda coba prediksi dalam ML yang diawasi. Ini juga disebut sebagai variabel hasil. Misalnya, dalam pengaturan manufaktur, variabel target bisa menjadi cacat produk.

daftar tugas

Alat yang digunakan untuk melacak kemajuan melalui runbook. Daftar tugas berisi ikhtisar runbook dan daftar tugas umum yang harus diselesaikan. Untuk setiap tugas umum, itu termasuk perkiraan jumlah waktu yang dibutuhkan, pemilik, dan kemajuan.

lingkungan uji

Lihat lingkungan.

pelatihan

Untuk menyediakan data bagi model ML Anda untuk dipelajari. Data pelatihan harus berisi jawaban yang benar. Algoritma pembelajaran menemukan pola dalam data pelatihan yang

memetakan atribut data input ke target (jawaban yang ingin Anda prediksi). Ini menghasilkan model ML yang menangkap pola-pola ini. Anda kemudian dapat menggunakan model ML untuk membuat prediksi pada data baru yang Anda tidak tahu targetnya.

gerbang transit

Hub transit jaringan yang dapat Anda gunakan untuk menghubungkan jaringan Anda VPCs dan lokal. Untuk informasi selengkapnya, lihat Apa itu gateway transit dalam AWS Transit Gateway dokumentasi.

alur kerja berbasis batang

Pendekatan di mana pengembang membangun dan menguji fitur secara lokal di cabang fitur dan kemudian menggabungkan perubahan tersebut ke cabang utama. Cabang utama kemudian dibangun untuk pengembangan, praproduksi, dan lingkungan produksi, secara berurutan.

akses tepercaya

Memberikan izin ke layanan yang Anda tentukan untuk melakukan tugas di organisasi Anda di dalam AWS Organizations dan di akunnya atas nama Anda. Layanan tepercaya menciptakan peran terkait layanan di setiap akun, ketika peran itu diperlukan, untuk melakukan tugas manajemen untuk Anda. Untuk informasi selengkapnya, lihat Menggunakan AWS Organizations dengan AWS layanan lain dalam AWS Organizations dokumentasi.

penyetelan

Untuk mengubah aspek proses pelatihan Anda untuk meningkatkan akurasi model ML. Misalnya, Anda dapat melatih model ML dengan membuat set pelabelan, menambahkan label, dan kemudian mengulangi langkah-langkah ini beberapa kali di bawah pengaturan yang berbeda untuk mengoptimalkan model.

tim dua pizza

Sebuah DevOps tim kecil yang bisa Anda beri makan dengan dua pizza. Ukuran tim dua pizza memastikan peluang terbaik untuk berkolaborasi dalam pengembangan perangkat lunak.

U

waswas

Sebuah konsep yang mengacu pada informasi yang tidak tepat, tidak lengkap, atau tidak diketahui yang dapat merusak keandalan model ML prediktif. Ada dua jenis ketidakpastian:

ketidakpastian epistemik disebabkan oleh data yang terbatas dan tidak lengkap, sedangkan ketidakpastian aleatorik disebabkan oleh kebisingan dan keacakan yang melekat dalam data.

tugas yang tidak terdiferensiasi

Juga dikenal sebagai angkat berat, pekerjaan yang diperlukan untuk membuat dan mengoperasikan aplikasi tetapi itu tidak memberikan nilai langsung kepada pengguna akhir atau memberikan keunggulan kompetitif. Contoh tugas yang tidak terdiferensiasi termasuk pengadaan, pemeliharaan, dan perencanaan kapasitas.

lingkungan atas

Lihat lingkungan.

V

menyedot debu

Operasi pemeliharaan database yang melibatkan pembersihan setelah pembaruan tambahan untuk merebut kembali penyimpanan dan meningkatkan kinerja.

kendali versi

Proses dan alat yang melacak perubahan, seperti perubahan kode sumber dalam repositori.

Peering VPC

Koneksi antara dua VPCs yang memungkinkan Anda untuk merutekan lalu lintas dengan menggunakan alamat IP pribadi. Untuk informasi selengkapnya, lihat Apa itu peering VPC di dokumentasi VPC Amazon.

kerentanan

Kelemahan perangkat lunak atau perangkat keras yang membahayakan keamanan sistem.

W

cache hangat

Cache buffer yang berisi data terkini dan relevan yang sering diakses. Instance database dapat membaca dari cache buffer, yang lebih cepat daripada membaca dari memori utama atau disk.

data hangat

Data yang jarang diakses. Saat menanyakan jenis data ini, kueri yang cukup lambat biasanya dapat diterima.

fungsi jendela

Fungsi SQL yang melakukan perhitungan pada sekelompok baris yang berhubungan dengan catatan saat ini. Fungsi jendela berguna untuk memproses tugas, seperti menghitung rata-rata bergerak atau mengakses nilai baris berdasarkan posisi relatif dari baris saat ini.

beban kerja

Kumpulan sumber daya dan kode yang memberikan nilai bisnis, seperti aplikasi yang dihadapi pelanggan atau proses backend.

aliran kerja

Grup fungsional dalam proyek migrasi yang bertanggung jawab atas serangkaian tugas tertentu. Setiap alur kerja independen tetapi mendukung alur kerja lain dalam proyek. Misalnya, alur kerja portofolio bertanggung jawab untuk memprioritaskan aplikasi, perencanaan gelombang, dan mengumpulkan metadata migrasi. Alur kerja portofolio mengirimkan aset ini ke alur kerja migrasi, yang kemudian memigrasikan server dan aplikasi.

CACING

Lihat menulis sekali, baca banyak.

WQF

Lihat AWS Kerangka Kualifikasi Beban Kerja.

tulis sekali, baca banyak (WORM)

Model penyimpanan yang menulis data satu kali dan mencegah data dihapus atau dimodifikasi. Pengguna yang berwenang dapat membaca data sebanyak yang diperlukan, tetapi mereka tidak dapat mengubahnya. Infrastruktur penyimpanan data ini dianggap tidak dapat diubah.

Z

eksploitasi zero-day

Serangan, biasanya malware, yang memanfaatkan kerentanan zero-day.

kerentanan zero-day

Cacat atau kerentanan yang tak tanggung-tanggung dalam sistem produksi. Aktor ancaman dapat menggunakan jenis kerentanan ini untuk menyerang sistem. Pengembang sering menyadari kerentanan sebagai akibat dari serangan tersebut.

bisikan zero-shot

Memberikan LLM dengan instruksi untuk melakukan tugas tetapi tidak ada contoh (tembakan) yang dapat membantu membimbingnya. LLM harus menggunakan pengetahuan pra-terlatih untuk menangani tugas. Efektivitas bidikan nol tergantung pada kompleksitas tugas dan kualitas prompt. Lihat juga beberapa bidikan yang diminta.

aplikasi zombie

Aplikasi yang memiliki CPU rata-rata dan penggunaan memori di bawah 5 persen. Dalam proyek migrasi, adalah umum untuk menghentikan aplikasi ini.

Terjemahan disediakan oleh mesin penerjemah. Jika konten terjemahan yang diberikan bertentangan dengan versi bahasa Inggris aslinya, utamakan versi bahasa Inggris.